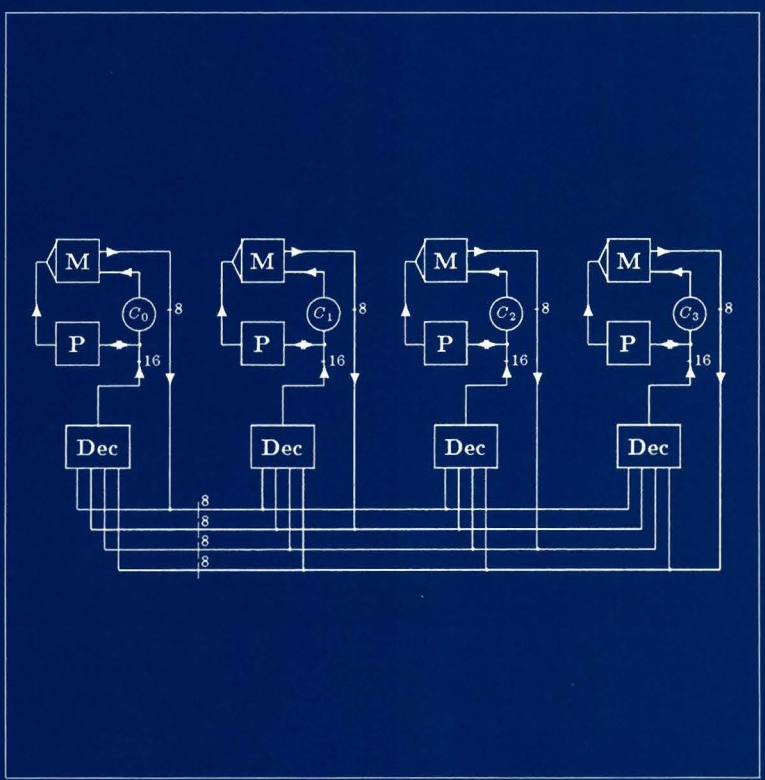
A generalization of fault-tolerance based on masking

A Generalization of Fault-Tolerance Based on Masking

Thijs Krol



A Generalization of Fault-Tolerance Based on Masking

Cover
The architecture of a (4,2)-concept fault-tolerant computer

Aan Anneke, Ingrid, Rinze en Eelco

A Generalization of Fault-Tolerance Based on Masking

Proefschrift

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. dr. J.H. van Lint, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op dinsdag 24 september 1991 om 16.00 uur.

door

Thijs Krol

geboren te Leeuwarden

Dit proefschrift is goedgekeurd door de promotoren:
Prof. ir. A. Heetman
en
Prof. dr. ir. J. Vytopil

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

Contents

Samenvatting

Het doel van dit proefschrift is het generaliseren de methode *masking* die wordt gebruikt ter verbetering van de betrouwbaarheid van digitale systemen. Tevens wordt een methode gepresenteerd die het correct functioneren van een fouten-tolererend systeem onafhankelijk maakt van een onbetrouwbare omgeving. Voor dit laatste doel is een nieuw en efficient algoritme voor interactieve consistentie ontwikkeld.

Tot nu toe werden fouten-tolererende digitale systemen veelal beschreven door te verklaren hoe in een bepaalde architectuur de extra, d.w.z redundante, onderdelen of deelsystemen worden benut om de betrouwbaarheid van het totale systeem te verbeteren.
In deze context dient betrouwbaarheidsverbetering te worden beschouwd als de verhouding tussen de Mean-Time-Between-Failures van een fouten-tolererend systeem en de Mean-Time-Between-Failures van een niet-fouten-tolererend systeem met dezelfde functionaliteit.

Fouten-tolererende architecturen worden tegenwoordig met succes toegepast in telefoon centrales, in computers voor het betaalverkeer, ruimtevaart, en zelfs in de burgerluchtvaart.
Het nadeel van de huidige ontwerpmethoden voor fouten-tolererende digitale systemen is het feit dat de betrouwbaarheid niet alleen afhankelijk is van de verbetering die verkregen wordt door het toepassen van een basisarchitectuur, maar dat de betrouwbaarheidsverbetering tevens afhankelijk is van allerlei ontwerpdetails. Bovendien wordt de betrouwbaarheidsverbetering bepaald door de vraag of het architectuurconcept consequent is toegepast of dat bepaalde aanpassingen zijn gemaakt teneinde de kosten te verlagen. Voor veel ontwerpen van fouten-tolererende digitale systemen blijkt het erg moeilijk te zijn om gedurende het ontwerpproces de ontwerpbeslissingen te

herkennen die kritisch zijn voor de betrouwbaarheidsverbetering. In feite komt het er op neer dat als gevolg van de toegepaste ontwerpmethode de betrouwbaarheidseigenschappen niet kunnen worden geverifieerd tijdens het ontwerpproces. Dus de uiteindelijke betrouwbaarheidsverbetering van het fouten-tolererend systeem hangt sterk af van de kwaliteit van het ontwerp, het ontwerpproces en het validatieproces. In ieder geval is de berekende of geschatte betrouwbaarheidsverbetering van veel ontwerpen twijfelachtig.

In dit proefschrift wordt beschreven hoe voor de klasse van fouten-tolererende digitale systemen die gebaseerd zijn op "masking" een aantal van deze nadelen weg te nemen zijn door het fouten-tolererend systeem te reduceren tot een verzameling gekoppelde Moore machines en door de kritische data communicaties in een dergelijk systeem te identificeren.

Bovendien zal het klassieke masking concept worden gegeneraliseerd door de meerderheidsfunctie, die de kern vormt van deze methode, te vervangen door een decodeerfunctie van een foutencorrigerende code. Het resultaat hiervan zullen we "Generalized Masking" noemen.

De totale klasse van fouten-tolererende digitale systemen, die gebaseerd is op Generalized Masking met inbegrip van de klassieke masking-systemen kan, indien de systemen op een voldoend hoog nivo van abstractie beschreven zijn, door twee deelklassen worden gekarakteriseerd, afhankelijk van de wijze waarop de deelsystemen verbonden zijn. Deze twee deelklassen worden gekarakteriseerd door respectievelijk twee functies,nl. $\mathcal{X}$ en $\mathcal{Y}$ of door drie functies nl. $\mathcal{X}, \mathcal{Y}$ en $\mathcal{Z}$, alsmede door het maximum aantal T te tolereren defecte modules. De systemen die door deze twee klassen beschreven worden zijn respectievelijk $(\mathcal{X}, \mathcal{Y}, T)$ systemen en $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ systemen.

De functie $\mathcal{X}$ beschrijft de manier waarop ieder van de modules in het fouten-tolererend systeem zijn informatie ontvangt van de buitenwereld. We zullen aantonen dat deze functie altijd correct moet worden uitgevoerd ook wanneer die buitenwereld foutieve en misleidende informatie naar het systeem stuurt. Is aan deze voorwaarde niet voldaan, dan kan het fouten-tolererend systeem zich incorrect gaan gedragen zelfs wanneer minder modules defect zijn dan volgens het ontwerp is toegestaan. Dit wordt het "Input Problem" genoemd.
Het correct uitvoeren van de functie $\mathcal{X}$ houdt in dat de correct functionerende modules in het fouten-tolererend systeem allemaal tot dezelfde con-

clusie moeten komen betreffende de informatie die zij vanuit de externe bron hebben ontvangen. Wanneer die externe bron correct functioneert dan moet die conclusie overeenkomen met de data die door die externe bron was verstuurd.

Algoritmen die soortgelijke eigenschappen bezitten als de eigenschappen die vereist zijn voor de functie $\mathcal{X}$ zijn de zogenaamde algoritmen voor interactieve consistentie of Byzantijnse Generaals Algoritmen. Sinds 1978 worden deze algoritmen onderzocht en sindsdien zijn vele resultaten gepubliceerd.

De algoritmen voor Interactieve Consistentie hebben het nadeel dat wanneer twee of meer fouten getolereerd moeten worden, een enorme hoeveelheid data tussen de modules van het fouten-tolererend systeem moet worden uitgewisseld. Voor praktische toepassingen betekent dit dat niet meer dan drie of vier defecte modules in een systeem getolereerd kunnen worden. Teneinde de hoeveelheid data die verzonden moet worden te verminderen, wordt een nieuwe klasse van algoritmen voor interactieve consistentie gepresenteerd, die minder communicatie vereist indien het aantal te tolereren defecte modules kleiner is dan vier. Helaas wordt geen verbetering verkregen voor het meest eenvoudige algoritme, nl. het algoritme dat geschikt is voor vier modules waarvan er ten hoogste eén fout mag zijn.

Tenslotte worden in dit proefschrift een aantal methoden gepresenteerd die het Input Problem oplossen. Deze algoritmen zijn gebaseerd op algoritmen voor interactieve consistentie.

Samenvattend is het doel van dit proefschrift

- De generalisatie van de ver-N-voudiging methode, welke gebaseerd is op een gedistribueerde uitvoering van een foutencorrigerende code. Het resultaat noemen we "Generalized Masking".

- De definitie van twee klassen fouten-tolererende systemen die de klasse van systemen die gebaseerd zijn op Generalized Masking karakteriseren.

- De presentatie van een bepaalde architectuur, het (N, K)-concept die gebaseerd is op Generalized Masking en waarvan de voordelen in de praktijk zijn bewezen.

- De presentatie van een symboolcorrigerende code ten behoeve van het $(4, 2)$ concept, die naast de symboolfouten ook nog in staat is bit-fouten te corrigeren zonder dat daar extra redundantie voor nodig is.
- De definitie van het Input Problem van een fouten-tolererend systeem.
- De presentatie van een nieuwe klasse van synchrone deterministische algoritmen voor interactieve consistentie die gebaseerd is op meerderheidsbeslissingen en foutencorrigerende codes en die in practische toepassingen minder data communicatie vereist dan de bestaande synchrone deterministische algoritmen voor interactieve consistentie.
- De oplossing van het Input Problem met behulp van gelijksoortige algoritmen als de algoritmen voor interactieve consistentie.

Summary

This thesis attempts to generalize a particular method, called *masking*, which is used for improving the reliability of digital systems, such as computer systems. Moreover a method is presented which makes the proper functioning of a fault-tolerant system independent of an unreliable external world. For the latter purpose a new and effective interactive consistency algorithm is developed.

Thus far fault-tolerant digital systems are mostly described just by explaining how the spare (i.e. redundant) components or subsystems in a particular architecture are utilized for improving the overall system reliability.
In the present context the reliability improvement should be interpreted as the ratio between the mean time between failures of the fault-tolerant system and the mean time between failures of a non-fault-tolerant system with the same functionality.

At present many fault-tolerant architectures are successfully applied in real world systems, such as telephone exchanges, space vehicles, computers for transaction processing, etc., and even in civil aviation.
The drawback of the current design methods for fault-tolerant systems however is that reliability improvement not only depends on the improvement achieved by the application of some basic architecture, but that the reliability also depends on many design details and whether the architectural ideas are implemented straightforwardly or whether some adaptations have been made in order to arrive at a more cost-effective design. In many fault-tolerant designs, design decisions which are critical with respect to the reliability improvement, are very difficult to recognize during the design process. In fact it often turns out that due to the design methods applied, the reliability properties of the system cannot be verified during the design process. Hence the

quality of the design, the design process and the validation process heavily determines the final reliability of the fault-tolerant system. At least the calculated or estimated reliability improvement of many of the present designs is questionable.

In this thesis we will try to overcome some of these drawbacks for the class of fault-tolerant architectures which are based on masking, by reducing a fault-tolerant digital system which is based on masking to a set of coupled Moore machines and identifying the critical data transfers.

Moreover, the classical masking concept, the N-modular redundancy scheme, will be generalized by replacing the majority vote, which is the key of this method, by the decoder function of an error-correcting code. The result will be called "Generalized masking".

Provided the systems are described on a sufficiently high level of abstraction, the entire class of fault-tolerant systems based on generalized masking, the classical masking systems inclusive, can be described in two ways depending on the interconnection of the subsystem. One of these subclasses is characterized by two functions, i.e. $\mathcal{X}$ and $\mathcal{Y}$ and the number T of faulty modules that can be tolerated. The other subclass is characterized by three functions $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z}$ and the number T of faulty modules that can be tolerated. The systems described by these two classes are called $(\mathcal{X}, \mathcal{Y}, T)$ systems and $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ systems respectively.

The function $\mathcal{X}$ describes the way in which each of the modules of the fault-tolerant system receives its information from the outside world. We will show that this function must be performed "fault free" even if the outside world produces incorrect data, otherwise the fault-tolerant system might go down even if it contains less faulty modules than it is designed to tolerate. This will be called the "Input Problem".
A "fault free" performance of the function $\mathcal{X}$ means that the correctly functioning modules in the fault-tolerant system all must come to the same conclusion about what the external source has sent them. And if the external source functions correctly, this conclusion should be the data which were sent by the external source.

Algorithms with properties similar to those which are required for the function $\mathcal{X}$ are the so-called Interactive Consistency Algorithms or Byzantine

Generals Algorithms. They have been investigated since 1978 and many results have been published since then.

The interactive consistency algorithms suffer from the fact that if two or more faulty modules are to be tolerated an enormous amount of data has to be transmitted between the modules of the fault-tolerant system. In practice this means that no more than three or four faulty modules can be tolerated in a system. In order to reduce the amount of data which has to be transmitted, a new class of interactive consistency algorithms will be presented which is based on error correcting codes and which if the number of faulty modules is four or less, requires less data to be transmitted than the existing algorithms. Unfortunately no improvement is obtained for the most simple algorithm which runs on 4 modules of which at most one may be faulty.

Finally we will present a number of methods which solve the Input Problem. These methods are based on an algorithm similar to the interactive consistency algorithms.

In summary this thesis aims at

- A generalization of the N-modular redundancy scheme which is based on the distributed implementation of error-correcting codes, and which will be called generalized masking.
- A definition of the two classes of systems which characterize the systems that are based on generalized masking.
- The presentation of a particular architecture, called the (N, K)-concept, which is based on generalized masking and the feasibility of which is proved by application in a commercial system.
- The presentation of a symbol-error-correcting code to be used in the $(4, 2)$-concept, which in addition to symbol-errors is also capable of correcting bit errors without requiring extra redundancy.
- A definition of the Input Problem of a fault-tolerant system.
- The presentation of a new class of interactive consistency algorithms which is based on voting and coding, and which requires in most practical applications less data transfer than the existing synchronous deterministic interactive consistency algorithms.

- The solution of the Input Problem on the basis of algorithms similar to the interactive consistency algorithms.

Preface

The field of fault-tolerant computing is still rather new. This can be concluded from a still continuing discussion on definitions and a lack of standard literature in which the area is treated from a formal point of view instead of from a phenomenological point of view. Therefore a short introduction to the field of fault-tolerant computing and one of its most intriguing issues, the so-called "Input problem" is presented in Chapter 1. In this chapter subsequently the aspects which determine the reliability of a digital system are discussed, the relevant reliability criteria are defined, and a survey is given of the various methods and techniques which are available for improving the reliability of digital systems such as fault avoidance and fault-tolerance based on error detection, masking redundancy or dynamic redundancy.
Futhermore, in this chapter we will point out that the method to be used often depends on the required form of reliability (fail-safe, fault-tolerant, survivable without repair) and the degree of improvement to be achieved.
In a separate section the arguments are presented which support the opinion that repairable fault-tolerant systems should be implemented by means of masking redundancy, the latter being the subject of this thesis.
Finally the "Input problem" which is an integral part of any fault-tolerant system will be explained.

In Chapter 2 the well known N-modular redundancy scheme will be generalized to a class of systems which we will call "Generalized masking redundancy". As an introduction to Generalized masking first a particular architecture, called the (N, K)-concept, which belongs to this class will be presented. This new fault-tolerant computer architecture is based on a "distributed implementation" of a symbol-error-correcting code. The faults in this (N, K)-concept are masked by this error-correcting code instead of by a majority vote function which is the case in N-modular redundant systems.

To understand better the time dependency of a synchronous digital system we will model a synchronous digital system by means of the Moore model and relate time and space by unfolding time into space.

Using as a basis the unfolded representation of the N-modular redundancy scheme we will identify and discuss the critical data transfers. We will show that the broadcast of data and the voting on the results can be replaced by the encoder function and the decoder function of an error-correcting code respectively. This can be implemented for the I/O of the system as well as for the state of the system. This results in the definition of a $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant system and a $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant system. Real fault-tolerant systems based on generalized masking will be based on a mixture of both. The basic ideas behind Generalized masking could also be described in terms of a "distributed implementation of an error-correcting code" or in terms of "the encoding of physically implemented functions".

The (N, K)-concept is described in detail for $N = 4$ and $K = 2$.
It will be shown that symbol-error-correcting codes with additional bit-error-correcting capabilities make additional memory protection by means of bit-error-correcting codes superfluous and a newly designed symbol- and bit-error-correcting code for the $(4, 2)$-concept will be presented.

The systems described in Chapter 2 are all based on the assumption that the Input Problem is solved. In Chapter 5 this finally will be done on the basis of interactive consistency algorithms. Chapters 3 and 4 will be devoted to these interactive consistency algorithms.

In Chapter 3 the Byzantine Generals problem, which is also called the Interactive consistency problem, will be sketched based on its original description. Its relevant parameters will be discussed and the requirements which have to be fulfilled by an algorithm which solves the problem are defined. Thereafter a survey will be given of the existing literature and the results obtained so far.

In the second part of Chapter 3 a new class of algorithms will be defined which will be called Dispersed Joined Communication algorithms and which satisfy some properties which can be regarded as a more liberal version of the interactive consistency requirements.
Based on these Dispersed Joined Communication algorithms a new class of algorithms for reaching interactive consistency will be presented. This class

of algorithms is based on voting and error-correcting codes and meets both the $N \geq 3T + 1$ bound and the $K \geq T + 1$ bound.
The class of Interactive Consistency algorithms based on voting and error-correcting codes comprises:

- the class of algorithms based on voting published in the early eighties, which we will call the classical algorithms.
- a new class of algorithms based on voting which require considerably less data communication than the classical algorithms and which meet both the $K \geq T + 1$ bound and the $N \geq 3T + 1$ bound.

The class of algorithms described in Chapter 3 contains algorithms which require much less data communication between the modules than the existing synchronous deterministic algorithms. In order to compare the new algorithms defined in Chapter 3 with the existing synchronous deterministic algorithms two criteria will be defined, i.e.:

- the number of messages which needs to be transmitted between the modules,
- the minimum size of the original message.

For these criteria a number of relations will be derived which make it possible to calculate these figures. For a large number of practical examples the resulting figures are presented.
Although the number of messages in our new class of algorithms based on voting and error-correcting codes, increases exponentially with the number of faults which are to be tolerated and the number of messages in one of the algorithms published by Dolev grows polynomial with the number of faults which are to be tolerated, we will show that for practical applications the algorithms in the class of algorithms which is based on voting and coding are favourable.

In Chapter 5 a method is presented which makes the proper functioning of a fault-tolerant system independent of an unreliable external world. In other words, the solution to the Input Problem will be presented. This solution will be extended to a general solution for the interconnection of fault-tolerant systems.
The correctness of the behaviour of a fault-tolerant system depends among other things on the correct distribution of the data of unreliable I/O devices

over the modules of the fault-tolerant system. A malfunctioning system, whether it is fault-tolerant or not, should never defeat a correctly functioning fault-tolerant system, i.e a system which does not contain more faulty modules than it is designed to tolerate. In order to cope with this problem, in Chapter 5 interactive consistency of communicating fault-tolerant systems will be defined. Thereafter a number of interconnection methods and algorithms will be presented which satisfy the above-mentioned interactive consistency. These interconnection methods and algorithms are all based on interactive consistency algorithms. The implementation of such an algorithm for interactive consistency between communicating fault-tolerant systems is described in detail for the (4,2)-concept fault-tolerant computer system architecture.

Chapter 1

An introduction to fault-tolerant computing

Various methods can be used for improving the reliability of computer systems. The method to be used often depends on the required form of reliability (fail-stop, fault-tolerant, survivable without repair) and the degree of improvement to be achieved.
In this chapter a survey is given of the various methods and techniques available, with emphasis on those techniques that are based on the addition of supplementary (redundant) hardware.
The arguments are presented which support the opinion that a repairable system of which a reliability improvement is required of the order of 100 should be based on masking redundancy.
It will be shown that any fault-tolerant system sets special requirements for the function which distributes the data of an external unreliable source over the modules of the fault-tolerant system. This is the "Input Problem".

1.1 Introduction

Even in the development of the first electronic calculating machines, reliability played an important role. The great number of electron tubes used and their low reliability made it impossible to run a computer program lasting longer than a few hours. Even so, much time elapsed before any useful literature covering the field of reliable computer systems was published. Among the earliest works were those of Shannon (1948) and Hamming (1950) on

which redundancy and error correction were founded, and that of von Neumann (1956) which established the basis for the use of redundancy to mask defective components.

In the course of the sixties the topic of reliable computer systems was tackled systematically by companies such as IBM (System 360) and Bell (No.1 ESS), and by the aviation and astronautics industry, although at that time, the exchange of ideas had hardly got underway. Not until the seventies did an explosion of literature occur in the field of computer reliability ,[FTCS 71-90], [Siew. 82], [And. 79], [And. 81].

The present chapter surveys the various aspects of fault-tolerant computing, the present state of engineering and the application of engineering methods. First of all emphasis will be laid on the hardware available to us, then dynamic redundant systems will be compared to systems based on masking and finally the Input problem which is common to all fault-tolerant systems will be sketched.

1.2 Designing reliable systems

1.2.1 The various stages in the life of a system

The reliability of a computer system is determined by more than just its architecture and the reliability of its components. The entire time, from specification via design to the end of the period of use, is decisive for reliability and has to be taken into account. Faults can occur at all stages and must often be treated in different ways. Table 1.1 summarizes the various stages and their corresponding sources of faults as well as how these faults are currently detected.

Faults originating at one stage often only become apparent at a much later stage. A well-known example of this is the occurrence of design faults in an operating system which manifest themselves only after years of operation. Another is the occurrence of faults in the design of the timing, which are often revealed during production only if an unfavourable combination of otherwise properly operating components is used.

stages	sources of faults	detection techniques
specification and design	faulty and ambiguous specification and faulty algorithms	simulation audits
prototype	faulty algorithms, wiring faults, assembly faults, timing faults, defective components	testing
production and installation	wiring faults, defective components assembly faults	testing of system, diagnosis and built-in means of detection
utilization	defective components users' faults environmental influences	automatic diagnosis, preventive testing of system, and built-in means of detection

Table 1.1: *The sources of faults and their detection techniques in the various stages of the life of a system*

1.2.2 Fault classification

A fault in a computer system, in its widest meaning, is *a deviation in the behaviour of the system with respect to what the user expects of it.*

This definition is so wide in its scope as to be questionable. Yet we will base our discussions on this definition in order to indicate that fault-tolerant computing goes much further than the design of systems in which some defective components can be tolerated without loss of functionality.

Faults can be divided into two main groups, i.e.:

- Design faults and
- Hardware faults.

Design faults

These include all faults which, after repair, lead to a system which differs

from the previous system in respect of design.
Examples that can be mentioned are: incorrect specification, hardware design faults and faults in the software. Depending on the specification and the way we observe the system, i.e. from the point of view of the manufacturer or user, software faults by the user can also sometimes be considered as design faults.

Hardware faults

These include all faults caused by a physical defect, for example a defective gate or connection.
The way in which hardware faults manifest themselves can differ greatly.
In the case of a *permanent* fault the deviation is stable.
An *intermittent* fault becomes apparent not continuously but at irregular intervals.
A *transient* fault occurs only once and cannot be traced later on.
The way in which a fault becomes apparent depends on the level at which the system is considered, i.e. at the logic level, at the subsystem level or at the system level. A "stuck-at" fault at the logic level (permanent fault) may manifest itself at the system level as an intermittent fault. A design fault, for example in the software, may become apparent so rarely that for the user it cannot be distinguished from a transient fault.

1.2.3 Reliability criteria

The standards that the reliability of a computer system must meet depend greatly on the application. This can be explained with a number of examples.

The space shuttle has a mission time of only a few days. The failure of certain functions that are performed by the computer system aboard will put the crew at risk. The system cannot be repaired during the mission. Hence the decisive criterion here is the probability that the system will still be functioning correctly after a week. This probability must be practically one.
An unmanned space vehicle to another planet has a mission time of a few years. A probability of 70% that the computer system on board will then still be functioning may be acceptable.

The *reliability function* used above, i.e. the probability of survival as a

function of time is, however, insufficient to define the reliability of a telephone exchange. Here a high degree of *availability* is demanded, which is expressed by the condition that the system shall not be out of operation for altogether two hours in forty years. The availability thus is the fraction of time in which the system is not out of operation. Hence the availability is only a relevant criterion for repairable systems.

Other criteria which are used to express the reliability are the mean time between failures, MTBF, the mean time between down, MTBD, and the mean time to repair, MTTR. The *mean time between failures* is the expected value of the time which elapses between the moment the system is started up or a previous failure has been repaired and the moment at which a (subsequent) failure appears. Notice that faults which are automatically corrected by the system and which do not cause the loss of the functionality of the system are also taken into account.
The *mean time between down* is the expected value of the time which elapses between the moment the system is started up (possibly after a repair) and the moment at which the system loses its functionality due to the occurrence of a fault. Notice that in this case, faults which are automatically corrected by the system and which do not cause the loss of the functionality of the system do not influence the MTBD.
Similarly the *mean time to repair* is the expected value which elapses from system down to system up.

Thus an unavailability of two hours in forty years, for example, can be interpreted as a MTBD (meantime between down) of forty years with a MTTR (mean time to repair) of two hours, but also as a MTBD of one year and a MTTR of three minutes. The designer is free to decide how this availability condition must be interpreted.

For computers to be used in civil aviation, where their reliability directly determines the safety of the flight, a system failure rate (i.e. the reciprocal of the MTBD) of 10^{-10} /hour is acceptable. This is equal to the failure rate of a resistor. Given the small numbers of computers used in this application, it is impracticable for such a failure rate to be subject to experiments.

Fortunately in most cases the specifications are not too strict and the degree of reliability is determined by the aspects of costs: the question of how much can additionally be invested in reliability for the purpose of effecting a saving

in service costs for the supplier and a saving in loss of production for the customer if the computer system is defective. Here it must be remembered that for the customer the costs from loss of production may often be ten times the actual repair costs.

The effect of the measures to improve the system reliability is often expressed in terms of the reliability *improvement factor*. This is defined as the ratio between the MTBD of the fault-tolerant system and the MTBD of an equivalent system in which no measures have been adopted to increase the reliability.

It is important to determine the part and the function of the system to which the reliability requirements apply. By way of example consider the control of a telephone exchange where the loss of data corresponding to a given connection is acceptable if it does not happen too often. Failure of the computing function would mean that a connection can no longer be established. The computing function and the program store must therefore meet high reliability requirements, but the storage of the data recording a certain connection need not have such a high degree of reliability.
The opposite occurs in a computer used for salary administration. The failure of the computing function is acceptable, provided it is recognized in time and does not lead to irreparable faults. The loss of stored information, however, is completely inadmissible.

1.3 Methods to improve the reliability of computer systems

Improvement of reliability can be approached in two ways, namely by fault avoidance and by fault tolerance.

1.3.1 Fault avoidance

Fault avoidance is achieved by using established methods of design and established design rules, and by the use of the most reliable components possible. With this manner of working one can achieve a reliability improvement factor of 10 without exceptionally high costs. Moreover properly designed cooling, the prevention of hot spots, stable supply voltages, etc. can also lead to considerable improvement. In addition the "*learning curve*" plays

an important role in the avoidance of defects. This curve indicates the improvement in reliability as a function of the number of systems produced. A factor of two to five in reliability improvement is not uncommon in large series.

1.3.2 Fault tolerance

If fault avoidance does not yield a satisfactory result or if it is too expensive, something extra must be added (redundancy) in order to cancel out the influence of defective components or subsystems. Such redundancy can manifest itself in two ways, namely in extra time and in extra hardware. To detect a defect one can, for example, repeat a calculation and compare the result, i.e. redundancy in time. If the calculation is performed on two different machines, there can be said to be redundancy in hardware.

In the span from defect to repair a number of stages can be distinguished:

- fault detection,
- fault localization,
- reconfiguration,
- recovery and restarting, and
- repair.

Fault detection
A defect need not immediately lead to a logic fault and a logic fault does not always result in a wrongly performed function. Some time will therefore pass between the occurrence of the defect and the instant at which the fault is detected. In this time the fault can propagate through the system and damage data elsewhere. It is even possible that a function improperly performed owing to a defect will not be detected at all.

Fault localization
Once a fault has been detected, the site of the defect must be determined as accurately as possible, and this must be at least down to the level of an exchangeable unit. As a result of fault propagation and inadequate means of detection it is possible that the means of detection will provide insufficient information about the location of the defect and that diagnostic programs

will have to be run in order to localize the defect. In fact the behaviour of the system is then tested once more. The cause of a transient failure will thus not be found and the chance of localization of an intermittent fault is small. Systems in which diagnostic programs must be run to allow the reconfiguration are therefore poorly equipped to cope properly with this kind of faults.
If the location of the defective unit and the location of the data which have been corrupted due to the defect can be different, then in addition to localization of the defect damage assessment is also required.

Reconfiguration
Once the site of the defect has been determined, the defective unit can be switched off automatically or be replaced by a stand-by unit. If no spare unit is present, the result is a system with more limited possibilities or a smaller capacity. Often this is acceptable, because only a part of the functions fulfilled by the system must meet strict reliability requirements. These are referred to as systems based on *graceful degradation.*

Recovery and Restarting
The defect may damage data in the system, and this data must therefore be restored. In many systems this is done by regularly making a "back-up", which means that all data of the computer is stored, a number of status codes inclusive. By restarting from this data, one circumvents, as it were, the faults made. If too much data is lost the system must be restarted after the whole system has once more been loaded.

Repair
This can take place "on-line" as well as "off-line". For on-line repair special provisions must be made to prevent disturbance of the system and to reintroduce a repaired component in the system.

In the different types of fault-tolerant systems we may not always find all the stages described above. The literature [Siew. 78] distinguishes three types:

- systems based on *fault detection,*
- systems based on masking of faults - *masking redundancy* or *static redundancy,* and
- systems based on reconfiguration - *dynamic redundancy.*

Systems based on fault detection

These systems are characterized by the fact that only fault detection takes place on-line. The aim is to detect a fault as quickly as possible, in order to prevent functions being performed wrongly or data being lost.
Diagnosis, repair and restarting are left to the user. This kind of system is used where high reliability is demanded of the function to be carried out and where the availability of the function is of less importance. If the system is automatically shut down to a safe state after a fault has been detected, it is called a *fail-stop* system.

Fault detection techniques can be built in at all levels of the system, both in the hardware and in the software. Actually each operating system is constructed and provided with checks such that a hardware fault generally leads to the system shutting down without causing further damage. This is achieved by inspecting in software whether certain locations in the memory may be altered or only read, or by checking whether the result falls within a previously determined set of valid results (consistency checks). These software checks are in fact designed to detect software faults. Many faults caused by hardware defects are also detected, but usually only after some time and without information regarding the site of the fault. Between the occurrence of the fault and its detection a great deal of information may already have been lost and functions may have been performed improperly.

An advantage of software checks is that they detect certain software faults (design faults) which are not seen by hardware detection methods.
Software checks must be considered as redundancy in time, because they take up additional computing time.

Fault detection techniques performed by extra hardware are:

- Duplication
- Fault-detecting codes
- Self-checking logic
- Watch-dog timers.

Duplication

The most rigorous method of detection of hardware faults is duplication of

an entire computer. In this case both machines receive the same information and simultaneously perform the same task. Comparison of the results leads to an almost 100% certain fault detection. The question is at what level the results should be compared. In the case of a system where no requirements are imposed on the availability, and strict requirements are imposed on the correctness of the performed function, it suffices to compare the output of both computers.

In many telephone computers duplication finds application as part of a dynamically redundant system in which many processes take place simultaneously. Rapid detection is essential here and therefore each data transfer between processor and memory is compared.
Duplication in this way requires careful synchronization of the processors in both computers.
At the level of abstraction which is of interest for the duplication strategy, most general-purpose computers are not fully deterministic in their behaviour. This means that when two machines processing the same status variables are offered the same input data, it is possible for the results to come out of the machines in different sequences. One possible reason is that the time references of both machines are never exactly equal, but it is also possible that internal delays of the logic cause two independent processes to be performed by both machines in different sequences. Machines that are not deterministic cannot be used in a strategy of duplication.

Duplication at subsystem level is also used. An example of this is duplication of the ALU (arithmetic and logical unit)

Fault-detecting codes

Fault-detecting codes demand much less redundancy than duplication. The best known method is the use of the parity bit. The addition of a bit to a word ensures that the number of 'ones' in a word is always even. If the probability of a single bit fault is much higher than the probability of a multiple bit fault, then this is an effective and cheap method. Almost every defect in a memory leads to single bit faults in a word if the different bits of a memory word are stored in different chips.
Parity bits are used particularly for fault detection in memories and parallel data paths.

Many logic circuits have the property that in the case of a defect the logic

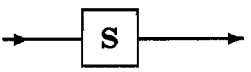
levels deviate only in one logic direction (unidirectional faults). This type of faults can be detected by means of M-out-of-N codes. Such codes consist of words of N bits in which the number of 'ones' is always M.

Check-sums

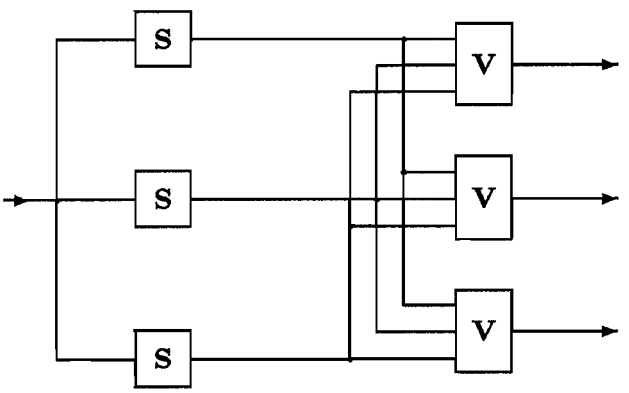
Check-sums are used very often to detect faults in tables. The best known method is the cyclic redundancy check. All check-sum methods boil down to performing a simple calculation on a table of numbers (data words). The result of the calculation, which is called the check-sum, is added as a redundant number (data word) to the table. If the table is read again, the same calculation is made again and the result is compared with the stored check-sum. If there is a difference, a fault must have occurred. The computing algorithm must be such that the most probable faults are always detected. The fault detection capacity for random fault patterns is very high. A random fault pattern is to be understood as arbitrary mutilation of the entire table. This high detection capacity is shown by the following. Suppose the results of the calculations of the check-sums are uniformly distributed between the numbers 0 and $Q-1$, i.e. the set of all possible tables can be divided into Q subsets such that all tables in a subset result in the same check-sum. The check-sum algorithm preferably should be such that all subsets are equally sized. In that case the probability that a random fault is not detected is 1 in Q. Thus if $Q = 2^{16}$, a redundancy of only 16 bits is necessary for achieving a probability of 99.998% for detecting random faults.

Self-checking logic

In the literature much has been written about self-checking logic. This is a logic circuit which supplies a redundant output, for example each logic output level is to be produced by two bits ("true" = 0 1 and "false" = 1 0). The circuit is then constructed so that a random defective gate will always lead (at one or more of the logic output levels) to the value 0 0 or 1 1. The problem with this technique is that it relies on only one of the gates in the network being defective. If all gates are integrated in an IC, this assumption applies to only a small percentage of possible faults. As a general rule the probability of detection is therefore too low. Nevertheless this kind of circuits are occasionally used.



single system or subsystem



triplicated system

S = system or subsystem
V = voter (majority decoder)

Figure 1.1: *Principle of triplication*

Watch-dog timers

Watch-dog timers monitor the time allocated to a certain function. If this time is exceeded, a fault report is generated. This kind of monitoring can be realized both in hardware and in software. A method widely used for inspecting the progress of the processes in a real-time system is the following: A hardware clock must for example be reset at least once each millisecond by the software. If this does not happen the computer is stopped. This method is effective because many hardware and software faults may cause the process to settle in a loop.

The effectiveness of a fault detection mechanism in a certain application is expressed as the *coverage*. This is defined as the probability that a (random) fault in a system is detected. According to this definition the coverage is determined in part by the frequency of occurrence of the different kinds of faults.

Systems based on the masking of faults

In systems based on fault masking, the tasks fault detection, fault localization and recovery form a whole. In fact the *computing process is not interrupted.* All data in the system must therefore be reproduced so that if, as the result of a defect, some of this data is incorrect, it will be seen from the rest of the data what it should have been.

The best known version is called triple modular redundancy TMR. In duplication we obtained fault detection by comparing the outputs. With triplication we are able to take a majority decision. It is important to triplicate these majority voters also, because otherwise they become decisive for the reliability of the system. Figure 1.1 represents the basic principle of TMR. The level at which the majority decision has to be taken depends on application.
One can, for example, allow three computers to perform the same functions synchronously and take a majority decision in respect of the results. A more reliable system is obtained by dividing the entire system into smaller modules, triplicating each module and providing it with a majority decider. The implementation of such a system, however, becomes more expensive because of the large number of voters.
It is also possible in the case of three computers to subject each data transfer from memory to processor to a majority decision. The architecture of such a system is presented in Figure 1.2. Here the input and output have been omitted. The only output of each module of the triad is the bus which sends the data via the voters to the processor. Irrespective of what goes wrong in a module, the fault will always be masked by the voters. This architecture can be considered as a feedback version of the method shown in Figure 1.1.

The voters will as a general rule be provided with additional hardware which records the fact that a fault has been masked and in which module it has occurred. This information need not be directly reported to the system

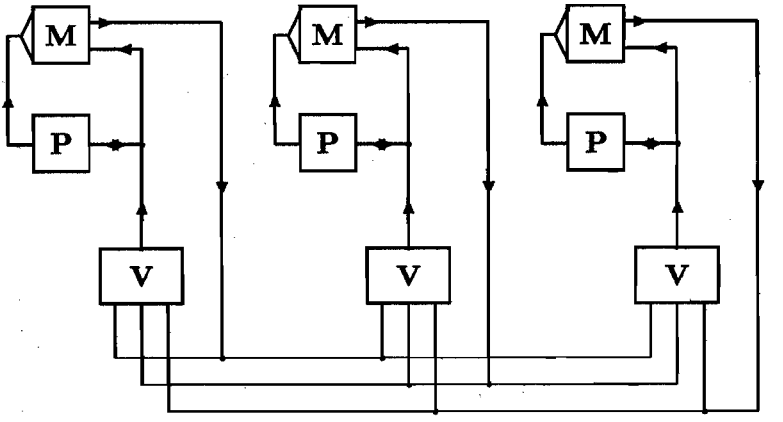


Figure 1.2: *A computer system triplicated at system level*

because it is of interest only to the user and is not necessary for a recovery process. This is in contrast to the systems which will be treated below and which are based on dynamic redundancy.

The addition of redundancy must take place very carefully. Extra hardware increases the probability of defects, with the possible consequence that reliability will decrease instead of increase. This effect occurs especially in non-repairable systems where the probability of survival as a function of the time is decisive.

If a defect causes a fault in more than one of the three modules, the system breaks down. The likelihood of this kind of fault must therefore be minimized, in other words the three modules must be completely independent. One speaks of *fault isolation areas* or *fault containment units* if the faults within such an area may be related but the faults in different areas are independent.

It is very difficult indeed to divide a system into fault isolation areas so that

the probability of related faults in different areas can be eliminated. Examples of these related faults are clock generators, supply voltages and causes lying outside the system, such as mains interference and strong electromagnetic pulses. The likelihood of faults by the user must not be forgotten either, for example pulling out a wrong board during on-line repair.

The great influence of these dependent faults will become apparent from the following numerical example. Let there be a threefold redundant system (TMR) consisting of three modules in which the amount of hardware will be three times that of a non-fault-tolerant system. The MTBF (mean time between failures) of a module will be the same as the MTBF of a non-redundant system; let it be 10^3 hour. In the non-fault-tolerant system any fault will cause the system to go down, hence the mean time between down, MTBD, of the non-fault-tolerant system is 10^3 hours.

Let the mean time to repair, MTTR, be 1 hour. Then the MTBD of the threefold system is 1.66×10^5 hours (about 20 years). The reliability improvement factor thus is 166.

This is shown by the following. The mean time between failures of each of the three modules of the threefold system is 10^3 hours. So on average every 333 hours a fault occurs in one of the modules. If such a fault occurs the mean time required to repair this fault is 1 hour. During the repair, the probability that a fault occurs in one of the two remaining correct functioning modules is 2.10^{-3}. Thus once per 500 occurrences of a first fault the system will go down due to a second fault in another module during the repair of the first fault. Hence the mean time between down of the threefold system is $500 \times 333 = 1.66 \times 10^5$ hours.

If 1% of the faults are dependent, so that they will cause a system crash, the MTBD becomes only 2.8×10^4 hours. This follows from the fact that after each first fault there is a probabilty of 10^{-2} that the second fault is a dependent fault which causes the system to go down and there is a probability of 2.10^{-3} that during repair of the first fault a second independent fault appears in one of the two remaining correct functioning modules. Hence the chance that a first fault leads to a system crash is 1.2×10^{-2}. Therefore the MTBD becomes $333/(1.2 \times 10^{-2}) = 2.8 \times 10^4$ hours.

The reliability improvement factor has thus fallen from 160 to 28.

An especially elegant and flexible solution results from multiplication of the processes in a multi-processor system and running the various copies of a process on different processors. [Wensley 78]. However *multiplication of*

hardware offers no protection whatsoever against design faults and software faults. So this solution only offers protection against hardware faults and therefore results in a very high availability only if both the hardware design and the software are relatively simple and their correctness can be proved.

Error-correcting codes

The use of error-correcting codes is a cheap and effective way of masking faults. If a data word is supplemented in a certain manner with redundant bits, then it is possible to correct one or more faulty bits. The percentage of redundant bits decreases with increasing length of the data word (assuming a constant correction capability) and the percentage of redundant bits increases if more bits require correction for the same length of the data word [MacW 78]. To protect an 8-bit data word against a one bit fault, four redundant bits are needed. For a 16-bit word this becomes only five bits. To allow correction of two bit faults simultaneously, about twice as many bits have to be added. The mutual dependence of the bit faults in one and the same data word can be made small by distributing the bits of a word over different chips. This fault masking method is currently used in many large memories.

Triplication can also be considered as an application of error correcting coding. After all, each data word is present in triplicated form in the system. The code word thus consists of three data words, two of which are redundant. The method presented in Figure 1.2 can therefore be considered as the application of an *error-correcting code at system level.* This can be generalized to (X,Y,T) or (X,Y,Z,T) fault tolerance, as will be explained in Chapter 2. The (N,K)-concept is a special implementation example of this generalization, and will also be discussed extensively in Chapter 2.

Systems based on dynamic redundancy

In the case of systems based on masking redundancy or dynamic redundancy, the functions of detection, localization, reconfiguration, recovery and restarting are performed automatically. The characteristic of dynamic redundant system is that the reconfiguration (and restarting) can be distinguished as a separate action (in the literature one often finds a somewhat differing definition).

The means available to us for fault detection have already been described

in Section 1.3.2. But in addition the more rigorous means described in 1.3.2, such as triplication, N-modular redundancy and (N,K)-concept fault tolerance, which will be dealt with in Chapter 2, can serve as a basis for dynamically redundant system. In that case the functions of detection, localization and part of the recovery form a whole, the advantage being that reconfiguration can be postponed to a more favourable moment.
We are thus able to distinguish two kinds of dynamically redundant systems:

- dynamically redundant systems based on the detection of faults, and
- dynamically redundant systems based on the masking of faults.

The disadvantage of the former is that transient and intermittent faults often cannot be localized. At the moment that the site of the defect is being sought by means of a diagnostic program, the defect will often have already disappeared. But by means of roll-back the fault can be cancelled. The influence on the reliability improvement factor of the poor ability of localization of transient and intermittent faults is difficult to determine.

Reconfiguration takes place by switching off the defective module and having its tasks taken over by another. The difficulty here is maintaining the integrity of the data.

1.4 Systems based on fault masking versus dynamic redundant systems

In the last two decades various methods have been proposed and implemented for improving the reliability and availability of computer systems by means of adding redundant hardware [Wensley 78], [Hopkins], [Gallager], [Siew. 82], [Avizienis], [Siew. 78]. Which of these methods is most suitable depends on the application and the required system reliability. This section, however, presents arguments which support our opinion that repairable fault-tolerant systems of which a reliability improvement is required to the order of 100, should be based on masking redundancy.

When a reliability improvement is required of less than ten, adding redundancy is not very cost effective. In such cases the reliability improvement can be achieved by fault avoidance, i.e. by improving the physical reliability of the components.

Dynamic redundant systems based on error detection are characterized by software-based recovery algorithms and in most cases they need diagnostic routines to trace the faulty module.
The recovery algorithms influence the application software in two ways:

- When a fault is detected the application program has to be interrupted immediately, which is not always allowed by the application.

- The recovery software always interferes with the application software and increases the complexity of the software design.

The penalty is very much higher software design costs and more difficult debugging of the software.
Because the software design is always the most complex part, this makes a case for implementing fault-tolerance in hardware.

The reliability improvement is strongly influenced by the concept of coverage, which was introduced by Bouricius [Bouricius]. Coverage is defined here as the probability that the system will recover given the existence of a fault. Let the coverage of a dynamic redundant system be c and let the mean time between down of a comparable non-fault-tolerant system be dn hours. In the non-fault-tolerant system each fault will result in a system crash. Hence the mean time between failures of the non-redundant system is also dn. The total amount of hardware needed for the redundant system clearly will be more than the amount of hardware needed to build the non-fault-tolerant system. Thus the mean time between failures dr of the redundant system will be less than dn. After the occurrence of a first fault the redundant system might go down for two reasons. Firstly, the recovery mechanism could fail, the probability of such an event is $1 - c$. Secondly, during the repair of the first fault and after a successful recovery, a second fault might appear which causes the system to go down. Let this probability be p (under the condition of successful recovery after the first fault). Then the mean time between down ddr of the redundant system is $dr/(1-c+c.p)$. Thus because $dr < dn$, the reliability improvement factor ddr/dn is less than $1/(1-c)$.
So these simple calculations shows that in a repairable system the reliability improvement is bounded by the coverage c to $1/(1-c)$.

In dynamic redundant systems the coverage depends on the effectiveness of the diagnostic and recovery programs. Intermittent and transient faults

often defy these diagnostic and recovery programs. In practice it therefore appears that a required coverage of 0.99 is hard to obtain.

As far as repairable systems are concerned all these objections can be obviated by employing masking redundancy.

Masking can be applied at different levels, i.e. at the logic level (quading), at (sub-)system level (N-modular redundancy, TMR) and at the process level (SIFT).

Owing to dependent faults in LSI and difficult fault localization, masking at the logic level has to be rejected.

Masking at the process level might be an attractive solution but it requires a lot of software overhead. Moreover additional circuitry is needed for fault localization and debugging.

For repairable systems we therefore propose a redundancy scheme which is based on masking at the (sub-)system level.

The most obvious solution would be to use TMR in which the voting is implemented at the data transfer level, as is done in the C.vmp. [Siew. 78].

In the design of a fault-tolerant system based on masking at data transfer level, for instance a TMR system, the location of the voters in the system determines to a great extent the performance of the system and the amount of hardware needed.

A TMR system consists of three *fault isolation areas*. Within these fault isolation areas (henceforth called modules) faults may be interdependent, whereas they are assumed to be mutually independent between the areas. In such a system the three modules run synchronously and all data are triplicated in the system. The voters have to mask any failure in a single module. This can be implemented by a majority vote on the data transfers between processor and memory and possibly on the address information sent by the processor to the memory.

The simplest solution however is only to vote on the data which are transferred from the memory to the processor. (The I/O will be neglected for the time being.) The architecture of such a system already has been given in Figure 1.2 on page 26.

1.5 The Input Problem

Fault-tolerant systems will always be connected to other systems based on different methods for reliability improvement. In any case they will be connected to basically unreliable input devices.
The interconnection of these external sources to a fault-tolerant system has to be done very carefully.
Two communicating fault-tolerant systems must never defeat each other as long as they are both functioning well, i.e. in both systems no more failures should occur than they were designed to tolerate. Also data originating from a malfunctioning system, which is not included in the fault-tolerant system, must never cause the receiving fault-tolerant system to go down.

The fault-tolerant systems discussed in the remainder of this thesis are all based on masking redundancy. So we assume that the fault-tolerant systems can be divided into a number of fault isolation areas such that faults are mutually independent between these fault isolation areas. We therefore only have to discuss the faults that can occur during the information exchange between the modules or between a group of modules and the environment.

In redundant systems data originating from a particular module are always broadcast to at least a number of other modules. When transmitting data from one module to a number of other modules two fault models can be distinguished, i.e.:

- The data received by the modules is erroneous, but all modules receive identical data.
- The modules receive different data (some of which may be correct).

The first fault model will be called the *classical fault model*, the second will be called the *byzantine fault model*. The faults in the latter model are said to cause *broadcast errors*. A pictural explanation of the difference between both fault models is given in Figure 1.3.
In most fault-tolerant designs only the first type of fault is taken into consideration. The second type of fault, called *broadcast errors*, cannot be ignored however. Experiments and practical experience have shown that it is even predominant in the event of power failures. The existence of broadcast errors is obvious when there is a separate connection between any two modules, but

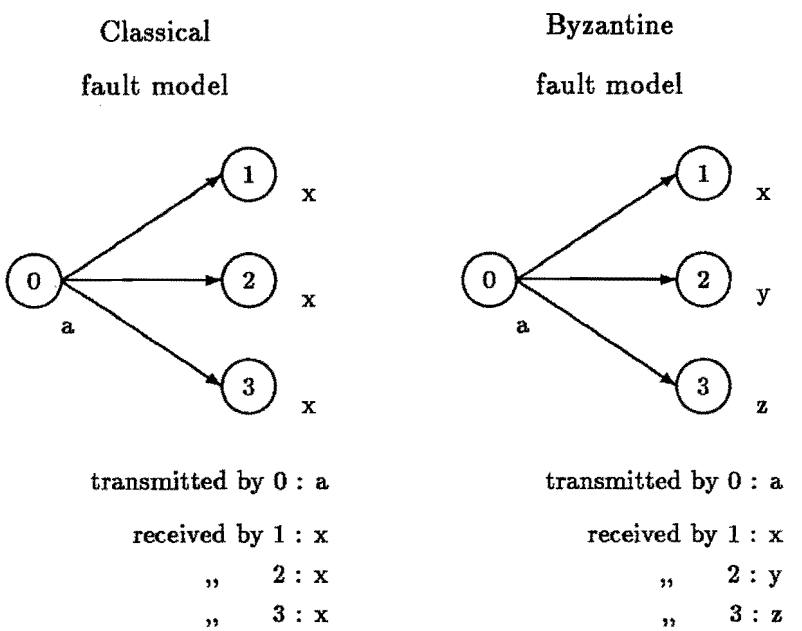


Figure 1.3: *The classical fault model compared to the byzantine fault model*

they also occur when a bus-type interconnection is used. In the latter case they are typically caused by timing failures or by failing bus drivers sending ambiguous logic levels. Notice that no two logic discrimination levels nor any two sampling instants are exactly identical.

The basic fault mechanisms which cause broadcast errors are elucidated in Figures 1.4 and 1.5. Figure 1.4 shows a broadcast error caused by a source sending an ambiguous logical level to two receiving modules which have different discrimination levels. Figure 1.5 shows that a source which changes its output due to a timing failure at the common sampling time causes a broadcast error. This is because receiver 1 samples the data just before and receiver 2 samples the data just after the trailing edge of the output signal of the source.

If an external faulty module produces broadcast errors, a fault-tolerant system which is still correctly functioning and which receives data from this module

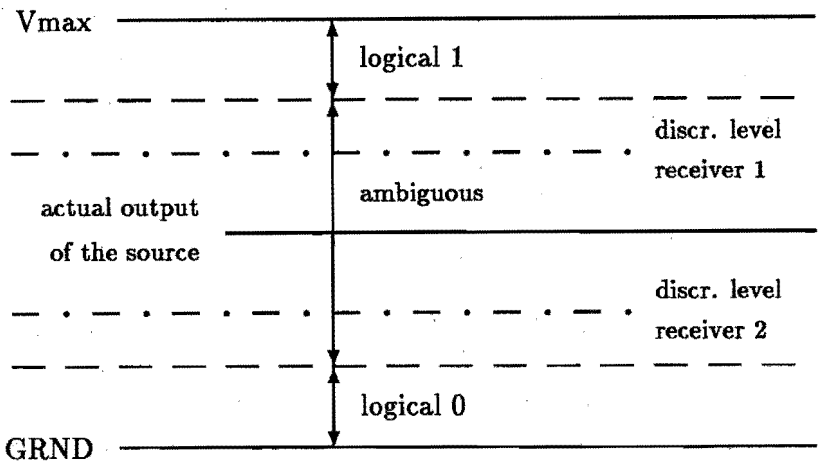


Figure 1.4: *Broadcast error due to a failing driver in the source and different discrimination levels of the receivers 1 and 2. The result is that receiver 1 will decide in favour of 0 and receiver 2 will decide in favour of 1.*

can be brought down by these broadcast errors.

This is what we call the *Input Problem.*
Unfortunately any fault-tolerant system has to co-operate with single unreliable sources. The way in which a fault-tolerant system might go down due to broadcast errors will be clarified by the following example:
Consider a triplicated system (a TMR system) of which one module is failing (in our example this is module 1), thus the system as a whole is still functioning correctly. A single module which produces broadcast errors is connected to this system. The failing module in the triplicated system also produces broadcast errors. The data flow in the system is elucidated in Figure 1.6.

The external module sends message A to module 3 of the TMR system and message B to the modules 1 and 2. Thereafter the modules 2 and 3 broadcast the message correctly but the failing module 1 sends A to module 3 and B to module 2. After the majority vote has been calculated in each module, both correctly functioning modules will disagree such that module 3 concludes that the message A was sent and module 2 concludes that the B was sent. The result of module 1 is assumed to be X, because this module is faulty.

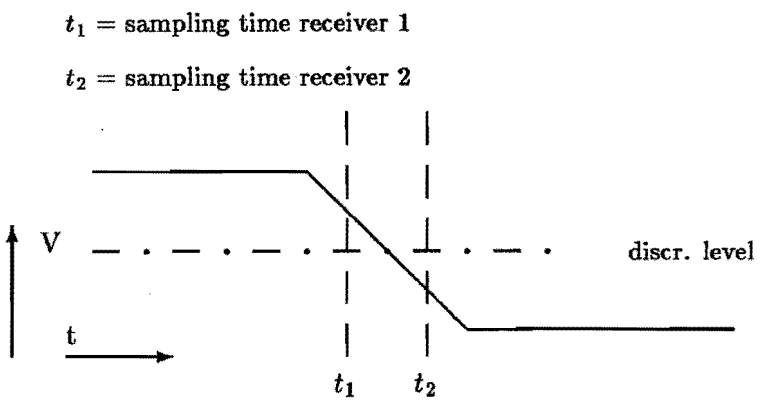


Figure 1.5: *Broadcast error due to a timing failure in the source and different sampling instances of the receivers 1 and 2. The result is that receiver 1 will decide in favour 1 and receiver 2 will decide in favour of 0.*

The TMR system will then go down because no correct majority vote can be obtained from the values X, B and A.

The Input Problem could be conquered by an algorithm which distributes the data from the external source over the modules of the fault-tolerant system and which has the following properties:

- The result of the algorithm is identical in all correct functioning modules of the fault-tolerant system, and
- if the external source is functioning correctly, i.e. it sends to all modules of the fault-tolerant system identical data, then the result of the algorithm in all correct functioning modules equals the data sent by the external source.

We will show in this thesis that such algorithms can be constructed under certain conditions.
The problem sketched above is related to the interactive consistency problem, which is also called the Byzantine Generals Problem. This problem is considered as one of the most important problems in distributed computing. The way in which this problem was originally formulated is presented

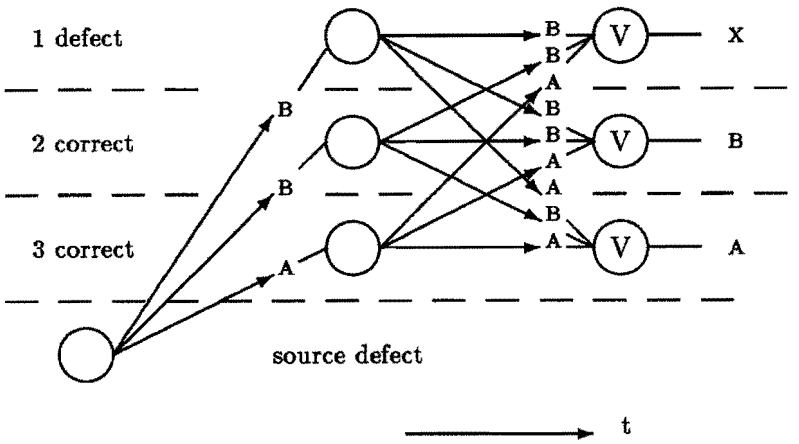


Figure 1.6: *The flow of data in a triplicated fault-tolerant system which receives data from a faulty external single source while one of the modules in the triplicated system is also defective. The result is a total system break down.*

in introduction of Chapter 3. Here the Byzantine Generals Problem will be defined in terms similar to the Input Problem, as follows:

Let there be N communicating modules with independent data links between the modules. Among these modules T or less are malfunctioning, probably transmitting conflicting information to different parts of the network, i.e. generating broadcast errors. Whenever one of the N modules, called the source, transmits a message to all other modules (or possibly conflicting messages when it is malfunctioning) by means of some algorithm, we say that this algorithm fulfils the *interactive consistency* requirements when the following conditions are fulfilled:

- The result of the algorithm is identical in all correct functioning modules and
- if the source is functioning correctly, i.e. it sends to all modules of the fault-tolerant system identical data, the result of the algorithm in all

correctly functioning modules equals the data sent by the source.

Algorithms which fulfil these properties are called Interactive Consistency Algorithms or Byzantine Generals Algorithms.

The similarity between the requirements which should be fulfilled by the algorithms which solve the Input Problem and the requirements for the Byzantine Generals Algorithms is obvious. The only difference lies in the fact that the Input Problem is based on an external source which may be faulty and a fault-tolerant system consisting of a number of modules of which some may be faulty, while the Byzantine Generals Problem is based on a number of modules in which the source is included.

Byzantine Generals Algorithms do exist in the case where $N \geq 3T + 1$. A literature survey of these algorithms is given in Chapter 3. Moreover in this chapter a new class of more efficient algorithms is presented.
The Byzantine Generals Algorithms are the basis of the algorithms which solve the Input Problem, [Krol 85], Chapter 5.

1.6 Conclusion

In the above sections we have considered the factors playing a role in the improvement of the reliability of a computer system and the techniques available for this purpose. Within this compass it is not possible to achieve completeness.

Which technique is to be chosen depends entirely on the kind and degree of reliability required. Even if this has been carefully specified in advance it still remains difficult to make a choice.
If a reliability improvement factor is allowed to be less than 10, there is little point in adding much redundancy. Such an improvement can be obtained by fault avoidance and a few simple measures in the hardware and in the software to mask the most serious faults.
If the reliability improvement factor needs to be of the order of 100, very rigorous methods will be required, preferably systems based on fault masking, or even dynamically redundant systems based on masking. The amount of redundant hardware to be added, together with the redundancy in time, will then quickly rise to 200 or even 300%.

The failure rate of the components from which a computer is constructed is reasonably well known. For that reason it is possible, if rigorous methods for the adding of redundancy are used, to calculate a kind of lower limit of the system reliability, *but only in respect of hardware defects.*

The "failure rate" of the designer will always remain the great unknown. Especially in the case of dynamically redundant systems based on fault detection this is a decisive factor. To determine the coverage of the system the designer must reason out all possible fault mechanisms and recovery strategies. The designer does not know what he has forgotten, and in order to achieve a reliability improvement of 100 the coverage must be better than 99.7%!

In systems with extremely high reliability, such as those required for critical aviation applications, credibility will also play an important role. Everybody is willing to believe that the probability of a wing breaking off from a plane in civil aviation is less than 10^{-10} per flying hour. But who will believe the designer of a computer system who claims that the MTBD of his system is 10^{10} hours, which is one million years! Yet this is a reasonable requirement in respect of a computer, the failure of which leads to the crashing of a plane.

Hitherto, for improvement of reliability, use was made as far as possible of software means. Often a consequence of this is that conditions are imposed on the designer of the users' software, such as the inclusion of check points, rollback etc. And in the case of real time control the designer must take into account that the normal operation of the system can be interrupted at any moment for, say, 0.1 second because a recovery action due to a defect is underway. This sometimes increases the complexity of the software quite considerably.

The ratio of hardware to software costs is shifting ever more in favour of the hardware. Hence expectations are that in the future ever more techniques based on hardware redundancy will be applied.
There are strong arguments which support the opinion that repairable fault-tolerant systems, of which a reliability improvement is required of the order of 100, should be implemented by means of masking redundancy.

The problem, the "Input Problem", of connecting external unreliable sources

to a fault-tolerant system such that a malfunctioning source cannot disturb the correct operation of the fault tolerant system has been solved. However there still is a need for more efficient algorithms.

Chapter 2

Generalized Masking Redundancy

This chapter describes a new fault-tolerant computer architecture based on a "distributed implementation" of a symbol-error-correcting code. In this, the (N, K)-concept as at is called, the faults are masked by this code.
The concept of "distributed implementation of an error-correcting code will be generalized to "the encoding of physically implemented functions", which will result in the definition of a $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant system and a $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant system.
The (N, K)-concept is described in detail for $N = 4$ and $K = 2$.
It is shown that symbol-error-correcting codes with additional bit-error-correcting capabilities make additional memory protection by means of bit-error-correcting codes superfluous and a newly designed symbol- and bit-error-correcting code for the $(4, 2)$ concept is presented.

2.1 Introduction

In Chapter 1 a number of arguments have been presented which support the statement that repairable systems of which the reliability improvement needs to be in the order of 100, should preferably be implemented by means of masking redundancy at the (sub)system level.
Thus far the only known example of masking redundancy at system level which can be applied for any digital system is N-modular redundancy. I.e. the comparable non-redundant system is N-fold implemented. Triple mod-

ular redundancy,TMR, thus is a special case with $N = 3$. At subsystem level masking redundancy is, in addition to N-modular redundancy, also implemented by means of error-correcting codes. However thus far error-correcting codes are only used for masking faults in memories and communication channels.

The purpose of this chapter is threefold, i.e.:

- To describe of a new fault-tolerant computer architecture, which will be called the (N, K)-concept. In this architecture error-correcting codes are applied at system level. The class of systems according to the (N, K)-concept can be considered as a generalization of the N-modular redundancy scheme.
- To identify the fundamental ideas behind masking redundancy on the basis of the combinatorial model and the Moore machine model and to define of two basic systems concepts which span the class of systems based on N-modular redundancy. I.e. N-modular redundancy with and without state voting.
- To generalize these two system classes by replacing the majority voters by the decoder functions of error-correcting codes.

In order to introduce the ideas behind generalized masking, in Section 2.2 the N-modular redundancy scheme is generalized to the new redundancy scheme, called the (N, K)-concept. In this section the meaning of faults, behaviour and correctness is still used in a rather sloppy way. The meaning of behaviour often leads to a lot of confusion, therefore in Section 2.3 we will first introduce and define two system models, i.e. the combinatorial model for combinatorial systems and the Moore model for sequential systems. Based on these models we will define the meaning of behaviour. In order to be able to talk about correct behaviour we need to compare the implementation with the specification. So the next step will be to introduce a meaning of specification, design and implementation which is suitable for describing fault-tolerant systems. Based on this a number of definitions of correct behaviour will be presented.
With these ingredients we are able to describe in Section 2.4 three classes of N-modular redundant (NMR) designs, viz. an NMR design of a combinatorial system, an NMR design of a sequential system without state voting and an NMR design of a sequential system with state voting.

The reliability properties of these classes will be discussed and it will be shown that NMR implementations can only be based on resettable systems.

In Section 2.5 the first two classes of the NMR systems mentioned above will be generalized to (X,Y,T) fault-tolerant systems and the last class, i.e. the one with state voting, will be generalized to (X,Y,Z,T) fault-tolerant systems. We will show that the reliability properties (X,Y,T) and (X,Y,Z,T) fault-tolerant systems are comparable with their corresponding NMR systems. Both classes will be elucidated with a number of examples and we will show that the (N,K)-concept is based on a mixture of both the class of (X,Y,T) and (X,Y,Z,T) fault-tolerant systems.

The Sections 2.6 to 2.10 are devoted to a detailed description of the (4,2)-concept. It is shown in these sections that symbol-error-correcting codes with additional bit-error-correcting capabilities make additional memory protection by means of a bit-error correcting code superfluous.
A new and optimal symbol- and bit-error-correcting code developed for the (4,2)-concept is presented in Section 2.7 and the design of its decoder is explained in Section 2.8. In Section 2.10 some facts about the implementation are presented.

2.2 The (N,K)-concept

As an introduction to the (N,K)-concept, we start with the TMR system described in Section 1.3.2 and 1.4.
Such a TMR system consists of three modules. Within these modules faults may be interdependent, whereas they are assumed to be mutually independent between the areas. The three modules run synchronously and all data are triplicated in the system. The voters have to mask any failure in a single module.
This is implemented by a majority vote on the data which are transferred from the memory to the processor. (The I/O will be neglected for the time being.) The architecture of such a system is given in Figure 2.1. The only output of a module, via which data can be sent to the other modules is the output of the memory. So whatever might go wrong in a single module it can only affect the memory output (possibly after some delay) and the fault will be masked by the voters.

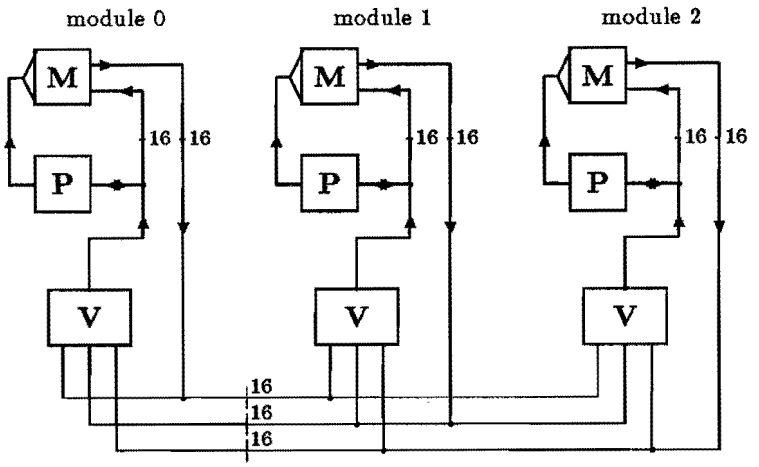


Figure 2.1: *An example of triple modular redundancy; the* $(3,1)$ *concept.*

A linear error-correcting code generally is defined as a set of n_c-tuples of symbols taken from some alphabet, [MacW 78]. The n_c-tuples are called the code words. The code words in the code are obtained by a linear mapping of the set of all k_c-tuples over the alphabet onto a set of n_c-tuples which constitutes the error-correcting code. The k_c-tuples are called the data words. In order to be able to correct T faulty symbols in a mutilated code word, the code words in the code should have a Hamming distance d_c such that $d_c \geq 2T+1$, i.e. any two different code words in the code differ in at least d_c symbol locations. Error-correcting codes are often characterized by a tuple (n_c, k_c) or a triple (n_c, k_c, d_c).
From any linear (n_c, k_c) error-correcting code a so-called systematic code can be derived in which at k_c symbol locations, the symbols in the code words are identical to the symbols in the corresponding data words. Hence k_c symbols in the code word form the data word. The remaining $n_c - k_c$ symbols are called check symbols.

In a TMR system the memory data as well as the processor data are tripli-

cated and are thus in fact encoded into a $(3,1)$ error-correcting code. This code consists of three symbols (the data words in the memories) one of which can be considered as the information symbol (the one-symbol data word) and the other two as check symbols. Because the code words of this code are a repetition of the data words, such a code is called a repetition code. The code is single symbol-error-correcting, i.e. an entire symbol may be mutilated. The $(3,1)$ repetition code thus is a particular (trivial) example of the class of (N,K) codes which consist of N symbols of which K symbols are information symbols and $N-K$ symbols are redundant check symbols.

The observation that in a TMR system the memory data as well as the processor data are encoded in a $(3,1)$ repetition code leads to the following generalization in which different coding schemes are applied for memory data and processor data.

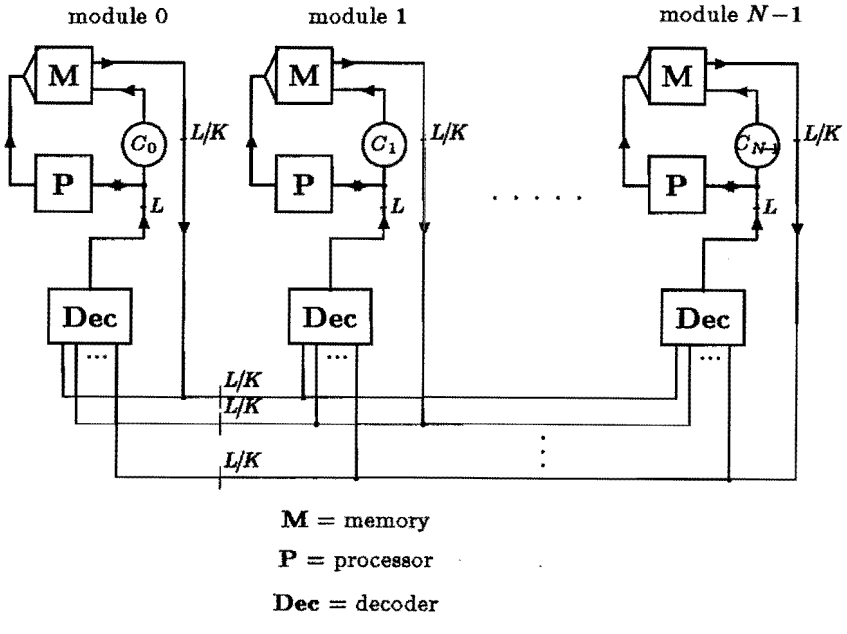


Figure 2.2: *Basic architecture of the (N,K)-concept*

In the (N,K)-concept shown in Figure 2.2 the processor data is N-fold repeated, i.e. encoded in a $(N,1)$ symbol-error correcting code. Thus in each module a copy of the processor data word is present. The memory data

however are encoded into an (N, K) symbol-error correcting code, where each of the modules contains one symbol of the code word.
Let the size of the data words of the processor in the comparable non-fault-tolerant system be L bits, then the processor data in the modules of the (N, K) concept is also encoded into L bit words. The symbol size, in terms of the number of bits to represent a symbol, of the (N, K) code is K times smaller than the size of the data word. So a data word of L bits is encoded into N symbols of L/K bits. Each module of the (N, K) concepts stores one symbol of the (N, K) code. So the size of the words stored in the memories is in the (N, K) concept K times smaller than the size of the words stored in the TMR system.

The applied (N, K) symbol-error-correcting code can be chosen to be Maximum Distance Separable (MDS) [MacW 78]. Such a code is capable of correcting up to $T = (N - K)/2$ randomly failing symbols and only requires a factor of $(N - K)/K$ additional memory hardware.

This is much less than the amount of additional memory hardware needed in the N-modular redundancy scheme. It has however to be paid for by a less efficient use of the redundant processor hardware.
The N-fold processor would in principle be capable of correcting
$(N - 1)/2$ randomly failing processors, but these capabilities are not fully utilized.

When data are transferred from the N processors $P_0, \ldots, P_{N-1}$ to the memories $M_0, \ldots, M_{N-1}$ (Fig.2.2) the N-fold data are encoded into N symbols such that there is no intercommunication between the N modules (fault isolation areas). The N partial encoders $C_0, \ldots, C_{N-1}$ together form an encoder of the (N, K) symbol-error-correcting code.

When data are transferred from the memories to the processors all modules receive the complete code word and in each module the decoder masks the faulty symbols. Notice that within a faulty symbol any number of bits may be wrong.

Because the only output of a module is a single symbol of the code word, the number of randomly failing modules that is tolerated by the system equals the symbol-error-correcting capabilities of the code.

Just as in the TMR system described in the previous section, no matter what goes wrong in a particular module it can only affect the output of the memory, probably after some delay.

Because the memory is in general the most unreliable part of the system, in most TMR systems the memory is protected additionally by a bit-error correcting code. Besides, it appears that due to the bit sliced implementation a bit-error-correcting code is very effective. Such a bit-error-correcting code requires in each memory 30 to 50% added redundancy, depending on the data word length. Therefore the amount of memory hardware in a TMR system is often more than four times the amount of hardware needed in a non redundant system.

In the (N, K)-concept the required bit-error-correcting capabilities can be combined with the symbol-error-correcting code without using additional redundancy. These codes are capable of correcting a number of bit-errors in different symbols which exceeds the number of symbols that can be corrected.

Initiated by the (N, K)-concept and the $(4, 2)$ single-symbol/double-bit error-correcting code described in Section 2.7, a large class of codes has since then been found, [Gils 86], [Gils 87], [Gils 88], [Boly-88].

As far as reliability improvement is concerned, an (N, K)-concept fault-tolerant computer is comparable with a $(N - K + 1)$-fold computer. Both are able to tolerate $(N - K)/2$ randomly failing modules.

When the location of some failing modules is already known to the decoders, i.e. some of the modules are already suspected due to preliminary knowledge of the fault behaviour of the system, the symbols descending from these modules can be considered by the decoders as erasures. An (N, K)-concept fault-tolerant system can tolerate simultaneously U symbol-erasures and T random symbol-errors as long as $2T + U \leq N - K$.

Hitherto in this chapter, for reasons of simplicity, masking redundancy has been described on the basis of a von Neumann computer. Although this is the most important application, in the next section, Section 2.5, masking redundancy will be placed in a broader context.

A more detailed description of the (N, K)-concept will be given in the Section 2.6, for the case $N = 4$ and $K = 2$. This description will be again based on a von Neumann computer, but the conclusions and the results can be easily generalized by the reader.

The (4,2)-concept fault-tolerant computer has been applied as a control processor in the SOPHO S2500 system, which is the fully digital business communication system produced by Philips, [Sopho].

2.3 On modelling the behaviour of fault-tolerant systems

In this section the notion of behaviour and correct behaviour will be defined on the basis of two system models, i.e. the combinatorial model and the Moore model.
In (fault-tolerant) digital systems we will be concerned with two types of circuits or subsystems from which these systems are built, i.e. combinatorial and sequential circuits. The name combinatorial conveys the idea that the output at any time is a function solely of the input at that time. However in reality the input to the system will always be provided before the output becomes available but both events take place during the same time slot. Sequential systems are those in which the current output does not only depend on the current input but also on a sequence of prior events.

2.3.1 The combinatorial model

In combinatorial systems the current output only depends on the current input. So the relation between the input value x and the output value y can easily be described by:

$$y = F_c(x) \tag{2.1}$$

The function F_c not only describes the way in which the input and output are related, but also the domain and codomain of the function, thus the value sets of the input and output. We will restrict ourselves to finite value sets.

In real systems a sequence of input values will be added to the combinatorial system resulting in a sequence of outputs. Such a sequence of values can be represented by a function on the time. We choose a finite set of time instances represented by the set $\mathbf{T}$.
So, let $\mathbf{T}$ be the finite set of time instances which are taken into consideration, such that $\mathbf{T}$ is a subset of the set of integers, $\mathbf{T} \subset \mathbf{Z}$, then the functions x and y representing the input stream and output stream are defined over the set $\mathbf{T}$ of time instances. If the codomains of x and y, i.e. the set of values which can be observed on the input and output, are denoted by $\mathbf{X}$ and $\mathbf{Y}$, then the types of x and y are determined by:

$$x \in (\mathbf{T} \rightarrow \mathbf{X})$$
$$y \in (\mathbf{T} \rightarrow \mathbf{Y}) \tag{2.2}$$

The relation between the input and output stream is then fully described by:

$$x \in (\mathbf{T} \rightarrow \mathbf{X})$$
$$y \in (\mathbf{T} \rightarrow \mathbf{Y})$$
$$y(t) = F_c(x(t)) \tag{2.3}$$

From an external point of view, a particular combinatorial system is fully specified by the description of the function F_c. Such a description includes the description of the domain, codomain and the way the input values are mapped on the output values.

2.3.2 The Moore model

Any system can be specified by a relation on the input and output streams of the system and the initialization. If the system output at a particular moment depends on the history of the input, i.e. on prior events, an easy way to cope with the history is to describe the system by means of the Moore machine model. In this model, the state at any time t is described as a function of the state and the input at the previous time instance $(t-1)$. The output is only a function of the momentary state. See for instance [Hill]. Thus:

$$y(t) = F'_o(z(t))$$
$$z(t) = F_s(z(t-1), x(t-1)) \tag{2.4}$$

With

$$F_o = F'_o \circ F_s$$

this set of equations is equivalent to

$$y(t) = F_o(z(t-1), x(t-1))$$
$$z(t) = F_s(z(t-1), x(t-1)) \tag{2.5}$$

In which $z(t)$ is the state at time t, and $x(t)$ and $y(t)$ are the input and output respectively. The state-function F_s and the output function F_o again not only specify the relation between the input and old-state on the one hand and output and new-state on the other hand, but also the domain and codomain. Thus F_o and F_s also specify the value-sets of input, output and state, i.e. the types.

Notice that the Moore model is very restricted, because:

- Time is discrete and only a finite number of time instances are taken into account. Thus the domain of the functions x, y, and z, are a linearly ordered finite set. This informally means there exists a first time instance called t_0 and a last time instance called t_l. After the time instance t_l we are no longer interested in the value which can be observed on the input, output and state.
- The value sets of the input, i.e. the codomain of the function x, the value sets of the output, i.e. the codomain of the function y, and the value sets of the state, i.e. the codomain of the function z, are all finite sets.

Our universe of discourse is thus a finite set, built from identifiable objects. This will hold throughout this thesis.

Notice that the Moore machine model differs from the Turing machine model, in the sense that the Moore model is much more restricted.

The graphical representation of a Moore machine is depicted in Figure 2.3.

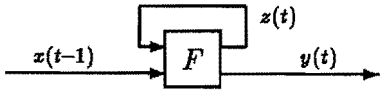


Figure 2.3: *The graphical representation of a Moore machine*

The functions x, y, and z, denoting the stream of input values, the stream of output values, and the successive state values respectively are not all defined

on the same domain.
Let $\mathbf{T}$ be the linearly ordered finite set of time instances which are taken into consideration,
let t_0 and t_l be the first and the last element in this set respectively,
let t_1 be the successor of t_0, and let $t-1$ be the predecessor of t provided a predecessor exists, then:
The function x is defined over the set $\mathbf{T}^-$ of time instances $\{t_0, t_1, \ldots, t_l - 1\}$.
The function y is defined over the set $\mathbf{T}^+$ time instances $\{t_1, \ldots, t_l - 1, t_l\}$.
And the function z is defined over the set $\mathbf{T}$ of time instances

$$\{t_0, t_1, \ldots, t_l - 1, t_l\}.$$

So if the codomains of x, y, and z, i.e. the set of values which can be observed on the input and output, and the state space, are denoted by $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Zz}$, then the types of x, y, and z are determined by:

$$\begin{aligned} x &\in (\mathbf{T}^- \rightarrow \mathbf{X}) \\ y &\in (\mathbf{T}^+ \rightarrow \mathbf{Y}) \\ z &\in (\mathbf{T} \rightarrow \mathbf{Zz}) \end{aligned} \tag{2.6}$$

From an external point of view, a particular system is fully specified according to the Moore model by the description of the functions F_o and F_s. Such a description encompasses the descriptions of the domains, codomains and the way input and old-state are mapped on output and new-state.

2.3.3 Unfolding time into space

In the Moore machine model the equations (2.5) specify the behaviour of the system recursively. This means that the relation between the input x and the output y is uniquely determined by the functions F_o and F_s provided $z(t_0)$ is known.

The difficulty of understanding the behaviour of sequential hardware lies in the notions of state and time. One possible way to circumvent this difficulty is unfolding the time into the space. This means that at each time instance a new instantiation exists of the (physically) implemented function. This principle is applicable to both the combinatorial model and the Moore model and is shown in Figure 2.4 and Figure 2.5 respectively. The result is a systolic array of identical timeless functions, in which the state is just an intermediate

variable. The time t in $x(t)$ has been reduced to just an index.

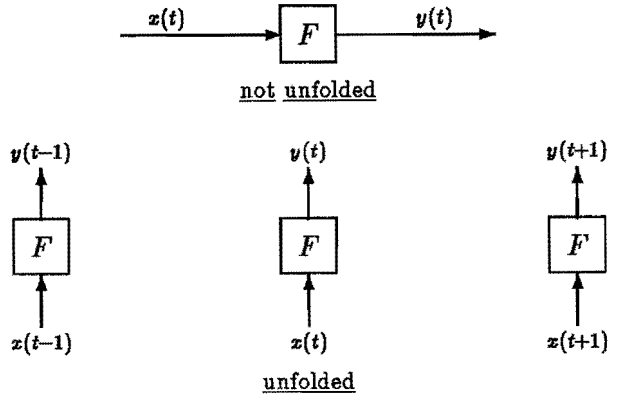


Figure 2.4: *Unfolding the time variable in the combinatorial model*

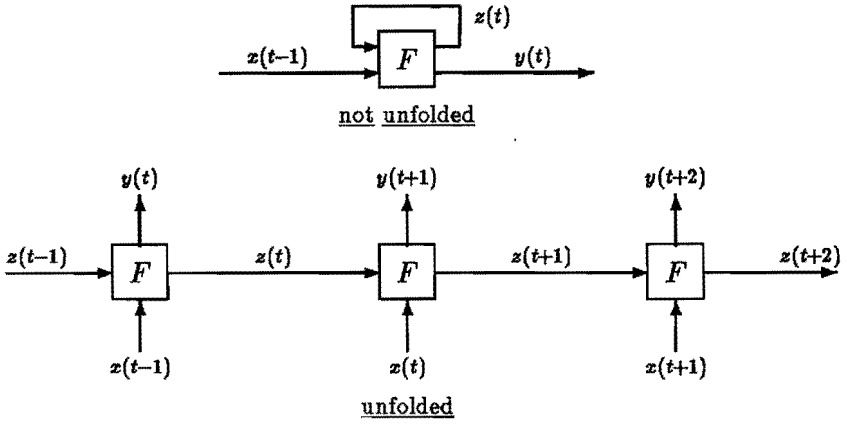


Figure 2.5: *Unfolding the time variable in the Moore machine model*

2.3.4 The meaning of behaviour

Talking about the behaviour of a system one generally refers to the relation between the input values and the output values. However, especially in the

case of sequential systems the notion of behaviour needs to be more carefully defined.

For this reason we will more carefully define the notions of

- behaviour, and
- behavioural equivalence.

Behaviour

In some cases and in particular in relation to real systems, behaviour is interpreted as a description of only the *observed* values at the terminals of the system. This interpretation does not suit our purpose. The term behaviour will only be used in the context of *specification, design* and *implementation.* Obviously specification and design are just models. But in this thesis an implementation also is only a model of what reality **could be**. So if we talk about the relation between the input and output values we mean all possible pairs of "input value, output value" which could or should be observed at the terminals of the implementation of the system. we thus use the mathematical meaning of relation, i.e. a subset of the cartesian product of the types of the input and output values.

So the behaviour of a specification, a design, as well as an implementation is expressed in the combinatorial model by a relation between the input values and the output values, i.e. the function F_c of the Combinatorial model, cf. equation (2.3).

In a sequential system the relation between the succesive input and output values depends on the state of the system, or at least on the initialization at the beginning of the period considered. Moreover further down in this section we also will have to treat malicious behaviour caused by faulty systems or subsystems. The correctness of system behaviour then will become time dependent. For this reason we will include the state in our definition of behaviour of a sequential system.

So the behaviour of a specification, a design, as well as an implementation is expressed in the sequential model by a relation between the input values and old-state values on the one hand and the output values and new-state values on the other hand, i.e. the functions F_o and F_s of the Moore model, cf. equation (2.6).

Because behaviour is described in the combinatorial case by the function F_c and in the sequential case by the functions F_o and F_s we conclude that behaviour of correctly functioning systems is independent of time.

Behaviour only relates to the values at the system terminals, and in the case of the Moore model also to the system state.
In general a system is always composed from a number of subsystems, modules or circuits. In a system description thus variables or signal values can be found which do not refer to the input, output or state at system level. These variables are called *internal variables.* If we want to discuss these signals we first have to decompose the system into subsystems. Then we are able to describe the properties of the variables which are internal at system level in the form of input or output variables at subsystem level. The properties of the variables which are internal at system level are then discussed in terms of the behaviour of a subsystem.

It is for this reason not possible to talk about the behaviour of a system before the model has been defined and before the terminals and, if appropriate, the state have been defined with this model. Neither is it possible to talk about the properties of internal variables without decomposing the system into subsystems.

Equivalence of behaviour

We say that two systems P_c and F_c which are described according to the combinatorial model are *behaviourally equivalent* if and only if

$$P_c = F_c$$

And, similarly, we say that two systems P_o, P_s and F_o, F_s which are described according to the Moore model are *behaviourally equivalent* if and only if

$$P_o = F_o \quad \text{and} \quad P_s = F_s$$

2.3.5 System decomposition

Fault-tolerant systems aim at diminishing the influence of faulty subsystems (modules). Therefore we need to decompose a system into subsystems.
Let the modules of a system be identified by the elements of a set Ns, then

the input, output and state at time instance t of a subsystem a will be denoted by $x(a)(t)$, $y(a)(t)$, and $z(a)(t)$ respectively, with $a \in \mathbf{Ns}$.
And thus

$$x \in (\mathbf{Ns} \rightarrow (\mathbf{T}^{-} \rightarrow \mathbf{X}))$$

$$y \in (\mathbf{Ns} \rightarrow (\mathbf{T}^{+} \rightarrow \mathbf{Y}))$$

$$z \in (\mathbf{Ns} \rightarrow (\mathbf{T} \rightarrow \mathbf{Zz}))$$

Similarly the function which specifies a module in the combinatorial model is denoted by $F_c(a)$ and the output function and the state function of a module a according to the Moore model are denoted by $F_o(a)$ and $F_s(a)$.

If a particular module a is behaviourally equivalent to some earlier defined module P (Moore machine) holds

$$F_o(a) = P_o \quad \text{and} \quad F_s(a) = P_s$$

In that case we say that module a is an *instantiation* of module P.

2.3.6 Specification, design and implementation

In the context of this thesis any system description is a model of a real or realizable system. However, a system can be described at different levels of detail. For this reason we will distinguish between *specification*, *design* and *implementation* as follows:

- The **specification** describes the (external) behaviour of the system. (what we wanted)
- The **design** describes in which way a system is composed from its subsystems together with the specification of these subsystems. (how we plan to make it)
- The **implementation** describes the real system. (what reality could turn out to be)

Obviously the external behaviour of a design can be derived from the design description. This external behaviour, in the case of a correct design, is equivalent to the one described by the specification.

Notice again that the description of the implementation is also just a model of what the real system could be.

This thesis aims at diminishing the influence of hardware faults. If we start by assuming that the specification and design are correct, we can only talk about faults in the implementation, i.e. a system implementation which does not satisfy the specification, or at subsystem level a subsystem implementation which does not satisfy the specification of the subsystem defined by the design.

2.3.7 Correct and malicious behaviour

Bear in mind that in the context of this thesis any system description is a model of a real or realizable system and that both the specification and the design are assumed to be correct.

Obviously the correctness of the implementation may be time dependent and thus it is useful to talk about correct and malicious behaviour at a particular time instance t or over a particular period of time.

Correct and malicious behaviour at system level

The combinatorial model

If the implementation is functioning correctly at time t then the values at the input and output at time instance t are in accordance to the specification. In other words the behaviour of the implementation at time t is equivalent to the behaviour expressed by the specification.

If the implementation does behave maliciously at time t, the relation between the values at the input and output may be different from the specification, but the phenomena at the terminals are still of the correct type.

Thus, if we let F_c be the specification of a system and let x and y represent the successive input and output values of the implementation, then:
A system which is behaving correctly at time instance t, with $t \in \mathbf{T}$, satisfies:

$$y(t) = F_c(x(t))$$
$$x(t) \in \mathbf{X}$$
$$y(t) \in \mathbf{Y} \tag{2.7}$$

and
A system which is behaving maliciously at time instance t, with $t \in \mathbf{T}$, *satisfies:*

$$x(t) \in \mathbf{X}$$
$$y(t) \in \mathbf{Y} \tag{2.8}$$

Consequently, correct behaviour is contained in malicious behaviour.

The Moore model

If the implementation is functioning correctly at time t the values at the output and the new-state of the implementation at time instance t must be related to the input and the old-state of the implementation at $t-1$ in accordance to the specification.
In other words at time t the behaviour of the implementation is equivalent to the behaviour expressed by the specification.

If at time t the implementation behaves maliciously, the relation between the values at the output and the new-state of the implementation and the input and the old-state of the implementation may be different from the specification, but the phenomena at the terminals and the states are still of the correct type.

Thus, if we let F_o, F_s be the specification of a system and let x, y, and z represent the successive input, output and state values of the implementation, then:
A system which behaves correctly at time instance t, with $t \in \mathbf{T}^+$, *satisfies:*

$$
\begin{aligned}
y(t) &= F_o(z(t-1), x(t-1)) \\
z(t) &= F_s(z(t-1), x(t-1)) \\
x(t-1) &\in \mathbf{X} \\
y(t) &\in \mathbf{Y} \\
z(t), z(t-1) &\in \mathbf{Zz}
\end{aligned}
\tag{2.9}
$$

and

A system which behaves maliciously at time instance t, with $t \in \mathbf{T}^+$, satisfies:

$$
\begin{aligned}
x(t-1) &\in \mathbf{X} \\
y(t) &\in \mathbf{Y} \\
z(t), z(t-1) &\in \mathbf{Zz}
\end{aligned}
\tag{2.10}
$$

Again, correct behaviour is contained in malicious behaviour.

Obviously if we are talking about correct behaviour over a period of time we mean correct behaviour at all time instances in that period.
From a practical point of view one could argue that this requirement is too stringent, because in fact we will only be interested in the external behaviour of the system, that is the relation between the input values and the output values over the period of time. Of course this relation depends on the state of the system at the beginning of this period and in the case of correct behaviour we possibly also wish that the state at the end of the period corresponds to the specification. However, the consecutive states during the period are in fact internal variables which may differ from the specification as long as they do not influence the relation between the input values over the period, the old-state at the beginning of the period, the output values over the period, and possibly the new-state at the end of the period.

We will however stay with the definition that in the case of correct behaviour over a period of time, at each time instance during that period the relation between the input, output and states is in accordance to the specification. The reason is that it will ease the analysis of fault-tolerant systems which

are going to be described in the next sections.

Correct behaviour should not be interchanged with correctness of the output values. At least according to the definition of behaviour presented in this thesis this is an incorrect interpretation. Correctness of behaviour here is only a proposition on the equivalence between the input, output and state relations expressed by the specification and the implementation. So in the combinatorial model we only may say that the output is correct at t if the input and the system are correct at time t. And in the Moore model we only may say that the output is correct at t if the system is correctly functioning at time instance t and the input and state are correct at $t-1$. So a system which is functioning correctly (at time instance t) may very well produce incorrect output values.

We will avoid talking about the correctness of output values as far as possible and if we do so it will be to be brief and always under the implicit assumption that the input and the state at the previous time instance are correct. So if we say that the output value is correct we always mean that the behaviour of the system which produces the output value is correct.

Correct and malicious behaviour at subsystem level

In a similar way the correct and malicious behaviour of modules (subsystems) can be described.
If we let $\overline{\mathbf{F}}_t$ be a set-function on $\mathbf{T}$ representing the module identifiers which refer to the modules which are functioning correctly at time t and let $\mathbf{F}_t$ be a similar set-function representing the module identifiers which refer to the modules which are functioning maliciously, such that for all t, with $t \in \mathbf{T}$, $\overline{\mathbf{F}}_t(t) \cup \mathbf{F}_t(t) = \mathbf{Ns}$ holds, then:
The correct and malicious behaviour at a particular time instance t of a particular module a is described by:

Combinatorial model

$$x(a)(t) \in \mathbf{X}$$

$$y(a)(t) \in \mathbf{Y}$$

$$a \in \overline{\mathbf{F}}(t) \Longrightarrow y(a)(t) = F_c(a)(x(a)(t)) \tag{2.11}$$

in which x, y, and z refer to the implementation and $F_c(a)$ refers to the specification of module a in the design.

Moore model

$$\begin{aligned} x(a)(t-1) &\in \mathbf{X} \\ y(a)(t) &\in \mathbf{Y} \\ z(a)(t-1), z(a)(t) &\in \mathbf{Zz} \\ a \in \overline{\mathbf{F}}(t) \Longrightarrow \big(y(a)(t) &= F_o(a)\big(z(a)(t-1), x(a)(t-1)\big) \\ \wedge\, z(a)(t) &= F_s(a)\big(z(a)(t-1), x(a)(t-1)\big)\ \big) \end{aligned} \tag{2.12}$$

in which x, y, and z refer to the implementation and $F_o(a)$ and $F_s(a)$ refer to the specification of module a in the design.

2.4 An abstract view on N-modular redundancy

In Chapter 1 the triple modular redundancy scheme, TMR, was used to elucidate fault-tolerant systems which are based on the masking of faults. This TMR scheme is just an example of the more general N-modular redundancy scheme (NMR) in which the "comparable" non-fault-tolerant system is N-fold implemented in order to tolerate up to $(N-1)/2$ failing modules.

At the level of abstraction which is sufficient for our discussions, the "comparable" non-fault-tolerant system may be regarded as the specification of the fault-tolerant design.

An N-modular redundant design system thus consist of N instantiations of its specification. Thus all N modules are identical.
In the following we will discuss the properties of three classes of NMR implementations, i.e.:

- An NMR implementation of a combinatorial system
- An NMR implementation of a sequential system without state voting
- An NMR implementation of a sequential system with state voting

2.4.1 An N-modular redundant implementation of a combinatorial system

An N-modular redundant design (NMR design) of a combinatorial system and its specification are depicted in Figure 2.6.

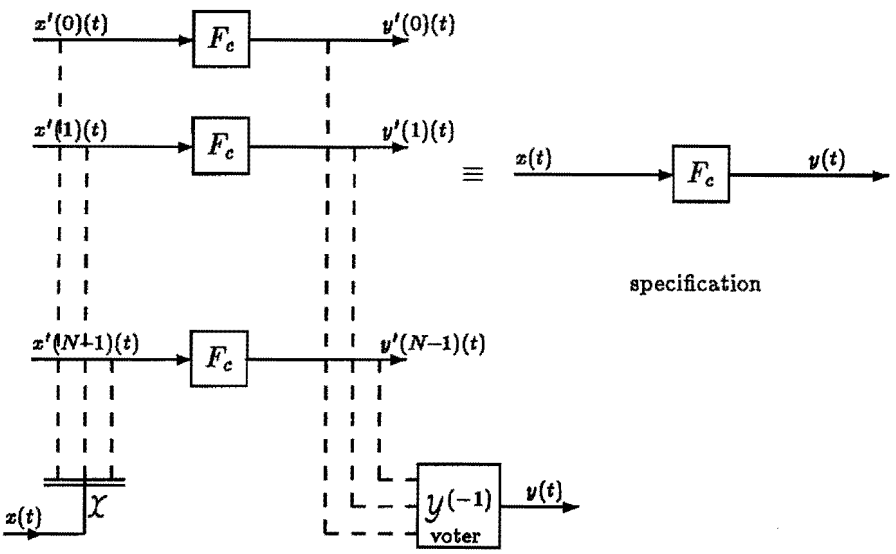


Figure 2.6: *A pictural representation of the relation between the design of an NMR combinatorial system and the specification*

In this picture the broadcasting of the input data is represented by a combinatorial function $\mathcal{X}$, which will be called the *distributing function*, and the majority-vote function is represented by the combinatorial function $\mathcal{Y}^{(-1)}$. The latter will be called the *observing function.*

The distributing function which is represented by the function $\mathcal{X}$ and the observing function which is represented by the function $\mathcal{Y}^{(-1)}$ are not a part of the NMR design.

The function $\mathcal{X}$ and the function $\mathcal{Y}^{(-1)}$ are only used to relate the behaviour

of the NMR design to the behaviour expressed by the specification. Including the function $\mathcal{X}$ and the function $\mathcal{Y}^{(-1)}$ in the design would suggest the existence of one or more modules which are not allowed to fail.
The correctness of the distributing function is related to the Input Problem which has been introduced in Chapter 1. The correctness of the observing function is left to the observer.

From Figure 2.6, the preceding definitions of behaviour and the fact that the majority-vote function $\mathcal{Y}^{(-1)}$, is able to mask the influence of $\lfloor \frac{N-1}{2} \rfloor$ faulty modules, follows that the behaviour expressed by an NMR implementation together with the distributing function and the observing function at a time t, equals the behaviour of the specification, provided that:

- at time instance t the behaviour of at least $(N+1)/2$ modules equals the behaviour expressed by the specification,
- at time instance t the behaviour of the broadcast function $\mathcal{X}$ is correct, and
- at time instance t the behaviour of the majority-vote function $\mathcal{Y}^{(-1)}$ is correct.

Implicitly assuming that the behaviour of the functions $\mathcal{X}$ and $\mathcal{Y}^{(-1)}$ is correct we may say that the behaviour of the NMR implementation is correct in the presence of T faulty modules at the same time, if $T \leq \frac{N-1}{2}$.
So even if each module behaves maliciously at one or more time instances, the behaviour of the NMR implementation still may be correct over all time instances t, with $t \in \mathrm{T}$. The only requirement is that at the same time instance at least the behaviour of $(N+1)/2$ modules is correct. Thus if the implementation satisfies:

$$\forall t \; : \; t \in \mathrm{T} \Longrightarrow |\overline{\mathrm{F}}_t(t)| \geq \frac{N+1}{2} \tag{2.13}$$

Another way to look at N modular redundancy is as follows:
The simplest error-correcting code is a $(N, 1)$ repetition code. The encoder of such a repetition code adds to any data word a code word which is composed by concatenating N words, each being identical to the original data word. Let the non-fault-tolerant system (the specification) be the data word, then

the N-fold system can be considered as a system which has been constructed from the non-fault-tolerant system by applying the encoder function of the repetition code. Notice that both the function and the data are encoded.
The N instantiations of the function F_c in the NMR implementation in Figure 2.6 are obtained by applying the encoder function of the $(N,1)$ repetition code on the function F_c in the specification. Also the data values in the NMR design follow from the data values in the specification by means of this $(N,1)$ repetition code.
By applying a decoder function (the observing function) on the results produced by the N functions the influence of faulty functions can be masked. The data produced by the correctly functioning modules provides sufficient information to derive the correct data.

The preceding illustrates clearly the principle of "encoding hardware". In the next sections it will be shown that encoding hardware can also be based on error-correcting codes other than the repetition code.

2.4.2 An N-modular redundant implementation of a sequential system without state voting

An NMR design of a sequential system and its specification are shown in Figure 2.7.
The behaviour of each of the modules in the NMR design is again identical to the behaviour expressed by the specification.
The broadcasting of the input data is represented in Figure 2.7 by a function $\mathcal{X}$ and the majority-vote function is represented by the function $\mathcal{Y}^{(-1)}$. Notice that both the function $\mathcal{X}$ and the function $\mathcal{Y}^{(-1)}$ are combinatorial functions. Moreover these functions again are not a part of the NMR design. The unfolded representation of the system in Figure 2.7 is given in Figure 2.8.

The correctness of the behaviour of the NMR sequential system will be judged by comparing the behaviour of the system consisting of the NMR system together with the functions $\mathcal{X}$ and $\mathcal{Y}^{(-1)}$, with the behaviour expressed by the specification.
The correctness of behaviour of an implementation is based on the equivalence of the behaviour of the implementation and the behaviour expressed by the specification, cf. (2.9) and (2.10) on page 58 . The input and output of

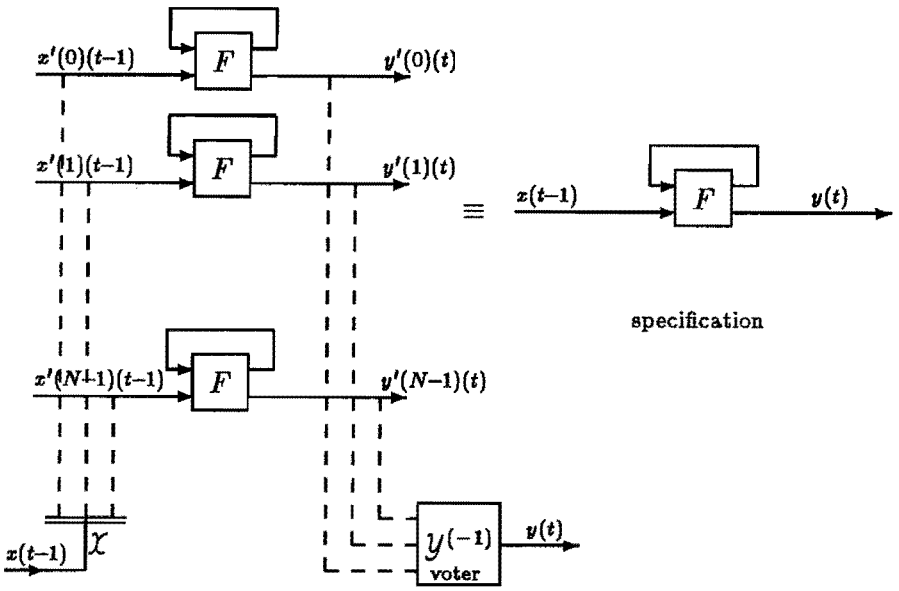


Figure 2.7: *A pictural representation of the relation between the design of an NMR sequential system and the specification*

the NMR system together with the functions χ and $y^{(-1)}$, are comparable with the input and output of the specification. In order to compare the old-state and new-state of the implementation with the old-state and new-state of the specification, we use the majority vote over both the old-states and the new-states of the modules of the implementation. This majority-vote function used for comparing the states of the modules in the NMR implementation with the state of the specification will be denoted by $S^{(-1)}$. The function $S^{(-1)}$ is intentionally not drawn in Figure 2.7, because neither in the NMR implementation itself nor in the environment of the system will the function $S^{(-1)}$ be found.

From Figure 2.8, the preceding remarks and the definition of behaviour at a time instance t, it follows that the behaviour at a time instance t of an NMR implementation together with the distributing and observing functions

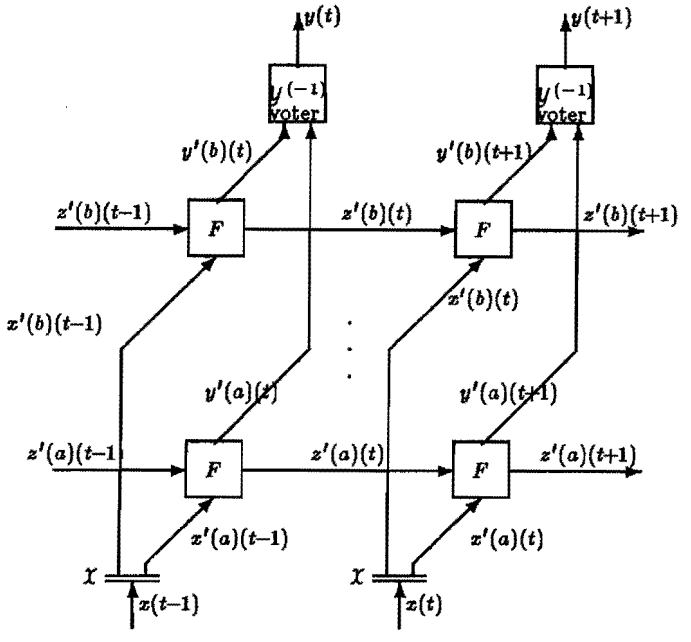


Figure 2.8: *The unfolded representation of a NMR sequential system without state voting, consisting of N modules; the modules are identified by a variable a, with $a \in \mathbf{Ns}$, but also by the variable t, because at each time instance t we assume a new instantiation of the module.*

equals the behaviour of the specification provided that:

- the behaviour of the broadcast function $\mathcal{X}$ is correct at time instance $t-1$,
- the behaviour of the majority-vote function $y^{(-1)}$ is correct at time instance t, and
- the behaviour at a time instance t, of at least $(N+1)/2$ modules equals the behaviour which is expressed by the specification,
- the state inputs $z'(a)(t-1)$ of all correctly functioning modules a are identical.

Notice that these conditions are sufficient but not necessary.

We implicitly assume that the first two requirements are fulfilled. And therefore if the last two requirement are fulfilled we say that the behaviour at time instance t of the NMR implementation is correct in the presence of T faulty modules, if $T \leq \frac{N-1}{2}$.

Notice that correctness of the behaviour at time instance t tells nothing about the correctness of the output value y at time instance t.

From the user's point of view we are only interested in the way the system responds to a particular input stream. Due to the requirement that all state inputs (old-states) of the correct modules must be identical, this definition might look rather unsatisfactory. Nevertheless it provides the basis for a detailed discussion of the properties of NMR sequential systems.

The requirement that all state inputs $z'(a)(t-1)$ of all correctly functioning modules a are identical is guaranteed if the modules that function correctly at time instance t are also functioning correctly at $t-1$ and are provided with identical state inputs $z'(a)(t-2)$. This by induction boils down to the sufficient requirement that:

- all modules which are functioning correctly at time instance t are also correctly functioning at all time instances prior to t and
- all modules are initialized at the same value.

This requirement however is still unsatisfactory, because in real systems a mechanism will be needed to force the state of the modules to a common value.

The problem of system initialization can be solved in two ways, i.e. with:

- systems with a limited history and
- systems which are resettable.

In some systems with a limited history, the output only depends on a limited number of prior inputs. This means that a number k exists such that for all t the output at t is independent of all inputs and states prior to $t-k$. Obviously in the Moore model k is at least one. If $k = 1$ we are dealing with a Moore model in which the state space consist of only one element. A simple example of a system with limited history is a delay line of k time units.

It is easily seen that if we are dealing with a system with a limited history k, the requirements for a correctly functioning NMR system may be reduced to

- over all periods of k successive time instances the number of modules which are functioning correctly at all time instances during that period, must be at least $(N+1)/2$.

NMR implementations of systems with a limited history of k time units which fulfil the previous requirement do not need to be initialized on a common value. They start behaving correctly after time instance $t_0 + k$.

In resettable systems one or more *reset sequences* are defined. A reset sequence consists of a number of well-defined consecutive input values which may be a part of the input stream. Reset sequences have the following properties. Let the length of a reset sequence be k. If the reset sequence is applied to the input of the system starting at t, then the state of the system at $t+k$ is independent of the state at t and thus independent from all states and input values prior to t. Notice that in the Moore model the input and old-state at t determine the output and new-state at $t+1$. Obviously the class of systems with a limited history are a subset of the class of resettable systems. This means that in the class of systems with a limited history k, any input sequence of length k is a reset sequence.

For resettable systems the requirements for a correctly functioning NMR implementation may be reduced to:

- over any period between two successive reset sequences, starting at the beginning of the reset sequence and ending at the beginning of the next reset sequence the number of modules which behave correctly at all time instances during that period, must be at least $(N+1)/2$.

NMR implementations of resettable systems which fulfil the previous requirement do not need to be initialized on a common value. They start behaving correctly after time instance $t_0 + k$, provided a reset sequence of length k is applied to the system starting at time instance t_0.

Summary of the properties of NMR implementations of sequential systems without state voting

- The NMR implementation without state voting is built from N copies

(instantiations) of the comparable non-fault-tolerant system (the specification).

- The system (specification) must be a member of the class of resettable systems.
- The distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ are not a part of the NMR system and are assumed to behave correctly.
- In the case of a resettable system:
  - We require that over any period between two successive reset sequences, starting at the beginning of the reset sequence and ending at the beginning of the next reset sequence, the number of modules which is functioning correctly at all time instances in that period must be at least $(N+1)/2$.
  - System initialization is obtained by means one of the reset sequences.
- In the case of a system with limited history k:
  - We require that over all periods of k successive time instances the number of modules which is functioning correctly at all time instances in that period must be at least $(N+1)/2$.
  - System initialization is automatically obtained because the NMR implementation starts behaving correctly after time instance $t_0 + k$.

2.4.3 An N-modular redundant implementation of a sequential system with state voting

The modules in a NMR implementation without state voting as described above are not interconnected. Or in other words, the different rows in the systolic array in Figure 2.8 do not influence each other. Consequently, a re-initialization of the system after a temporary defect in one of the modules or after on-line repair can only be done by interference from outside the system. Hence the reliability of the fault-tolerant system depends on something outside the system. The environment might wrongly reset the system. Moreover, in many systems the data stored in the memory needs

to be recovered after a reinitialization. This is impossible in these systems without the help of the environment.

This drawback can be overcome by letting the modules observe and influence each other.
In an NMR implementation without state voting, the observer outside the system observes at each time instance one column of output values and is able to derive by means of the function $y^{(-1)}$ from these data the correct output of the system in the presence of some faulty modules.
Similarly, at suitable time instances, (for instance each time instance), a module can observe the column of state variables and determine from it by means of a majority vote the correct value of the state variable in the presence of some faulty modules. This value is used for re-initializing the module. The majority vote function which operates on the state outputs of the modules will be denoted by $Z^{(-1)}$. We will call such a system an NMR *system with state voting.*

The pictural representation of an NMR system with state voting is shown in Figure 2.9. The unfolded representation of this design is given in Figure 2.10. In the latter picture the distributing function X and the observing function $y^{(-1)}$ are omitted. For simplicity we assume that the re-initialization takes place each time instance. Clearly the majority vote function $Z^{(-1)}$ is able to mask the influence of the same number T of faulty modules as the observing function $y^{(-1)}$ does.

In NMR designs with state voting the functions $Z^{(-1)}$ are a part of the NMR design and are prone to faults, but the functions X and $y^{(-1)}$ are not a part of the design and are considered to behave correctly in the same way as defined in the combinatorial case and the sequential case without voting.

In order to re-initialize each module automatically at each time instance, each module is provided with a voter to determine the correct state value. Malicious behaviour of such a state-voter at time t causes malicious behaviour at time t of the module in which the voter resides. Notice that in the Moore model the output of the voter is just an internal variable.

The correctness of the behaviour of the NMR sequential system with state voting will again be judged by comparing the behaviour of the system con-

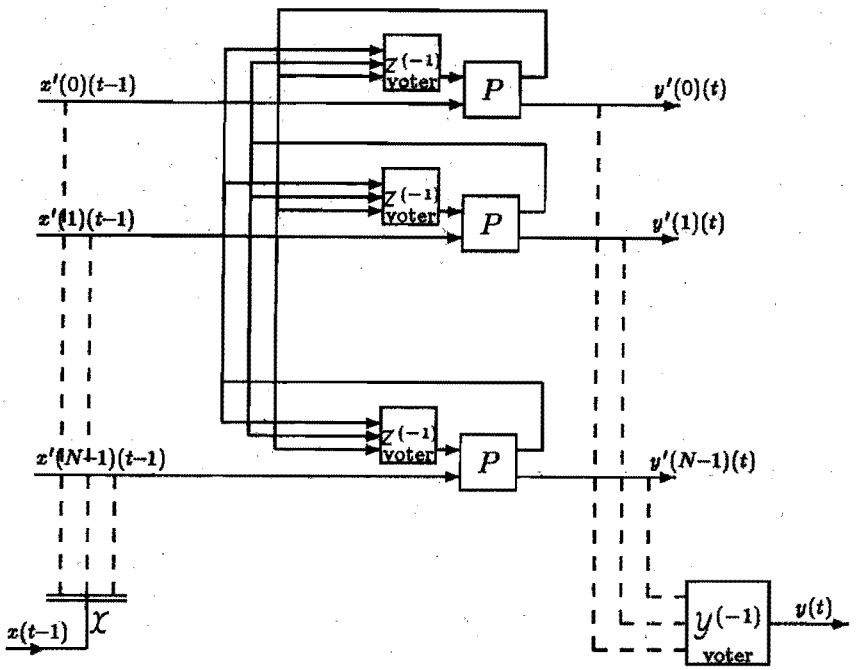


Figure 2.9: *A pictural representation of the of an N-modular redundant system with state voting*

sisting of the NMR system (of course including the state-voters) together with the functions χ and $y^{(-1)}$, with the behaviour expressed by the specification. Again the old-state and new-state of the specification are compared with the majority vote taken over the corresponding state of the modules.

From the Figures 2.9 and 2.10 and from the definition of behaviour at a time instance t, it follows that the behaviour at a time instance t of an NMR implementation equals the behaviour of the specification provided that:

- the behaviour of the broadcast function χ is correct at time instance $t-1$,
- the behaviour of the majority-vote function $y^{(-1)}$ is correct at time

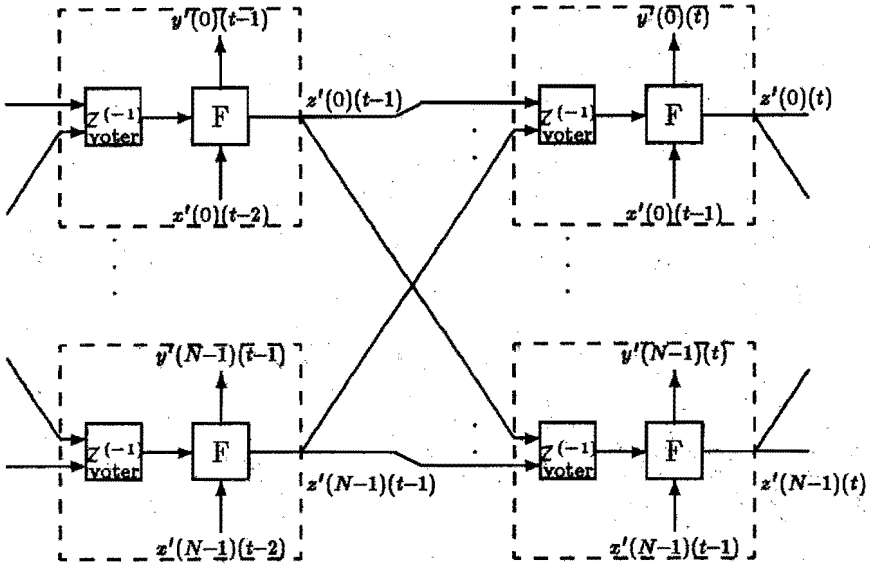


Figure 2.10: *The unfolded representation of an NMR design. The distributing function X and the observing function $Y^{(-1)}$ are omitted*

instance t and

- the behaviour at a time instance t of at least $(N+1)/2$ modules equals the behaviour which is expressed by the specification,
- at least $(N+1)/2$ the state outputs $z'(a)(t-1)$ are identical.

We implicitly assume that the first two requirements are fulfilled. And therefore if the last two requirements are fulfilled we say that the behaviour at time instance t of the NMR implementation is correct in the presence of T faulty modules, if $T \leq \frac{N-1}{2}$.

The requirement that at least $(N+1)/2$ state outputs $z'(a)(t-1)$ are identical is guaranteed if at time instance $t-1$ also at least $(N+1)/2$ modules are functioning correctly and are provided with at least $(N+1)/2$ identical state inputs $z'(a)(t-2)$. By induction this boils down to the sufficient requirement that:

- At time instance t and each time instance prior to t at least (N+1)/2 modules are functioning correctly and
- all modules are initialized at time instance t_0 at the same value.

So also NMR systems with state voting must belong to the class of resettable systems. The advantage of systems with state voting is the ability of each module to recover the state value after a failure. This makes online repair possible without the need to reset the entire system. Resetting an NMR system with state voting is only required with initial start-up of the system and after a complete system crash.
Like the NMR implementations without state voting and a limited history k, the implementations with state voting and limited history are automatically initialized, both at t_0 and after a complete system crash.

Summary of the properties of NMR implementations of sequential systems with state voting

- The NMR implementation is built from N copies (instantiations) of the comparable non-fault-tolerant system (the specification). To each of the modules a majority voter $Z^{(-1)}$ is added, which receives the state information from all other modules. The state input (old-state) of the copy of the specification is provided with the result of the function $Z^{(-1)}$.
- The system (specification) must be a member of the class of resettable systems.
- The distributing function X and the observing function $y^{(-1)}$ are not a part of the NMR system and are assumed to behave correctly.
- In the case of a resettable system:
  - We require that at each time instance t at least (N+1)/2 modules are functioning correctly.
  - System initialization at t_0 is obtained by means one of the reset sequences.
- In the case of a system with limited history k:

- We require that at each time instance t at least (N+1)/2 modules are functioning correctly.
- System initialization is automatically obtained because the NMR implementation starts behaving correctly after time instance $t_0 + k$.

2.5 A generalization of masking redundancy

2.5.1 Introduction

In the preceding we already pointed out that the distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ may be interpreted as the encoder and decoder function of an error-correcting code respectively. In this case it was an $(N, 1)$-repetition code. This observation suggest a generalization in which the distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ may be the encoder and decoder function of an error-correcting code which is less trivial than the repetition code.

First we will explain the basic principle of this generalization and thereafter we will explain two basic classes of fault-tolerant systems, i.e.:

- $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems which characterize the generalization of the NMR implementations of combinatorial systems and the NMR implementations of sequential systems without state voting.
- $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant systems which characterize the generalization of the NMR implementations of sequential systems with state voting.

The basic principle will be explained based on the combinatorial model and the most simple specification, i.e. the identity function (=). In Figure 2.11 the NMR design of such a system together with the distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ is visualized. The distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ are the encoder function and the decoder function of a $(N, 1)$-repetition code respectively. The decoder function thus is a majority voter.
Clearly if no more than $(N-1)/2$ of the (=) modules behave maliciously, the relation between x and y still satisfies the specification.

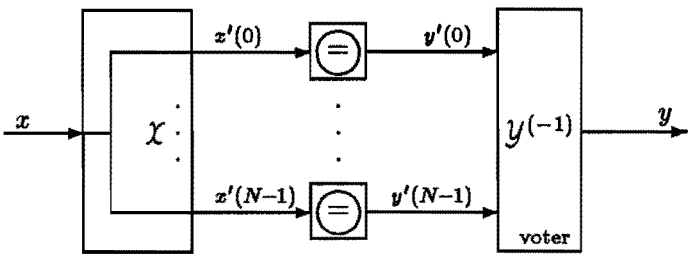


Figure 2.11: *A graphical representation of the N-modular redundancy scheme. The function X is a simple broadcast function. The function Y is a majority voter.*
If the broadcast function and the voter are functioning correctly and less than half the number of "=" boxes fails, then the relation between the input x and the output y is not affected by errors caused by the "=" boxes.

Figure 2.12 shows a fault-tolerant design of the identity function similar to the NMR design. However, in this design the distributing function X in Figure 2.11 has been replaced by the encoder function Y of a T-error-correcting code and the majority voter $Y^{(-1)}$ in Figure 2.11 has been replaced by the corresponding decoder function of the T-error-correcting code. Notice that the decoder functions $Y^{(-1)}$ in Figure 2.11 and Figure 2.12 are different functions. The encoding function Y in Figure 2.12 is divided into N partial encoding functions $Y(i)$ with $0 \leq i \leq N-1$, such that $Y(i)$ delivers the i-th symbol of the code word.
Because the code is able to correct T errors, the fault-tolerant implementation in Figure 2.12 behaves correctly, provided that, no more than T of the $(=)$ modules behave maliciously.

The encoder and decoder function of an error-correcting code are denoted by Y and $Y^{(-1)}$ respectively. Clearly

$$Y^{(-1)} \circ Y = Id$$

in which Id is the identity function.
Notice that $Y \circ Y^{(-1)}$ is not an identity function.

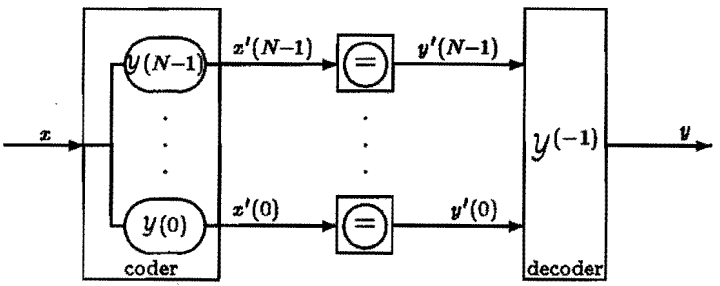


Figure 2.12: *A graphical representation of an encoder and decoder which protect the "=" boxes against failures. If the encoder and decoder are functioning correctly and the number of failing "=" boxes does not exceed the error-correction capability of the code, then the relation between the input x and the output y is not affected by errors caused by the "=" boxes.*

The part of the encoder function y, which produces symbol a of the code word, will be called the *partial encoder function for symbol* a and will be denoted by $y(a)$. So in Figure 2.12 it holds that

$$x'(a) = y(a)(x)$$

Remember that an error-correcting code usually is denoted by the triple (N, k_c, d_c), in which N denotes the number of symbols of which the code word is composed, k_c denotes the number of symbols of the data word and d_c is the Hamming distance of the code [MacW 78]. A code with Hamming distance d_c is able to correct T random errors if $T \leq \frac{d_c-1}{2}$. The Hamming distance of a code is at most $n_c - k_c + 1$. This bound can always be met provided the symbol size b_c is sufficiently large. Codes which meet this bound are called Maximum Distance Separable (MDS) codes. If we restrict ourselves for practical reasons to codes which are defined over some binary extension field, then the type of the variables may be expressed in the symbol size b_c, i.e. in the number of bits required to represent a symbol. In that case for $b_c \geq \log_2(N-1)$ a code which meets the $d_c = n_c - k_c + 1$ bound always exists.

In Figure 2.12 the size of the variables x and y thus is $k_c.b_c$ and the size of the variables $x'(i)$ and $y'(i)$ is b_c. The type of the indentity function performed in the modules thus differs from the type of the identity function which describes the specification.

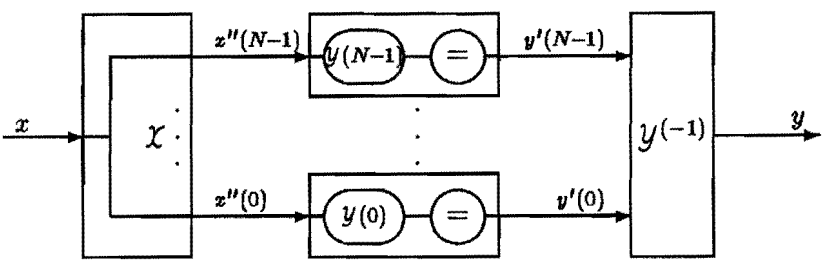


Figure 2.13: *A fault-tolerant system based on generalized masking. The distributing function X is a simple broadcast function. The observing function $y^{(-1)}$ is the decoder function of a T-error-correcting code.*
If the distributing function and the observing function behave correctly and the number of modules which behave maliciously does not exceed the error correction capability of the code, then the relation between the input x and the output y is equivalent to the specified identity function.

The distributing function and the observing function in the design shown in Figure 2.12 are the encoding and decoding function of the same T-error-correcting code y respectively. This however is not required. In Figure 2.13 a design is presented in which the distributing function X is the encoder function of a $(N, 1, N)$ repetition code, while the observing function $y^{(-1)}$ is the decoder function of an arbitrarily chosen T-error-correcting code with parameters (N, k_c, d_c). In order to get the behaviour of this fault-tolerant implementation including the distributing function and the observing function, equivalent to the specification, the partial encoding functions $y(i)$ of the encoder function y have been shifted from the distributing function to the modules which performed the identity function $=$.

In this fault-tolerant design, Figure 2.13, the size of the variables x and y

again is $k_c.b_c$ and the size of the variables $y'(i)$ is b_c, but the size of the variable $x''(i)$ is $k_c.b_c$.

From the Figures 2.12 and 2.13 it is easily seen that a fault-tolerant implementation according to Figure 2.13 behaves correctly if at least $N - T$ modules behave correctly.

The preceding discussion clearly shows that the NMR design can be generalized by replacing the distributing function and the observing function by the encoder and decoder functions of error-correcting codes respectively.

It is also not necessary that all modules in the fault-tolerant design are identical copies of the specification. The behaviour of modules may differ between modules and may be different from the specification.

In the previous section in which we elaborated on the different NMR implementations we identified three classes of NMR implementations, i.e.:

- NMR implementations of combinatorial systems,
- NMR implementations of sequential systems without state voting, and
- NMR implementations of sequential systems with state voting.

Apart from the distinction between combinatorial and sequential systems the first two classes are characterized by the distributing function $\mathcal{X}$, the observing function $\mathcal{Y}^{(-1)}$ and the number T of faulty modules which can be tolerated. The third class is characterized by $\mathcal{X}$, $\mathcal{Y}^{(-1)}$, T and the state voter $\mathcal{Z}^{(-1)}$. Therefore the generalization illustrated above can be divided into two classes, i.e.:

- $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems and
- $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant systems.

The first class applies to both combinatorial and sequential systems and the second class only applies to sequential systems.

2.5.2 $(\mathcal{X}, \mathcal{Y}, T)$ Fault-tolerant systems

$(\mathcal{X}, \mathcal{Y}, T)$ Fault-tolerant systems are built from N modules as is depicted in Figure 2.14 and are able to tolerate T faulty modules. The modules in the

design are identified by the elements of a set **Ns** and the behaviour of the modules is specified by the functions $Q(a)$ with $a \in$ **Ns**. The distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ are not a part of the fault-tolerant design but characterize the system and are needed to relate the behaviour which is expressed by the fault-tolerant design to the specification.

The correctness of the behaviour of $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems is based on comparing the behaviour of the implementation with the behaviour of the specification by means of the functions $\mathcal{X}$, $\mathcal{Y}^{(-1)}$, and $\mathcal{S}^{(-1)}$. This corresponds to the definition of correctness for NMR implementations. Recall from page 64 that $\mathcal{S}^{(-1)}$ maps the N-tuples of states onto the state expressed by the specification. The function $\mathcal{S}^{(-1)}$ is in general not a majority-vote function but may be any decoder function of an error-correcting code.

So a $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems can de defined as follows:

Definition 2.1 *A system belongs to the class of $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems if the behaviour of the fault-tolerant implementation together with the functions $\mathcal{X}$, $\mathcal{Y}^{(-1)}$, and $\mathcal{S}^{(-1)}$, in the presence of T or less faulty modules, is equivalent to the behaviour expressed by the specification.*

The unfolded representation of a $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant system is shown in Figure 2.15. From the definition of the class of $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems, the definition of NMR systems without state voting and a comparison of the Figures 2.14 and 2.15 with the Figures 2.7 and 2.8, it immediately follows that the reliability properties of $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant implementations are identical to those of NMR implementations without state voting and of which the number of modules is $2T + 1$.
Obviously the class of NMR systems without state voting is a subclass of the class of $(\mathcal{X}, \mathcal{Y}, T)$ systems.

2.5.3 Examples of $(\mathcal{X}, \mathcal{Y}, T)$ Fault-tolerant systems

If $\mathcal{X}$ is a broadcast function and $\mathcal{Y}^{(-1)}$ is the decoder function of a non-trivial T-error-correcting code, many practical examples of such a fault-tolerant system can be devised.

Let $\mathcal{Y}$ be the encoding function which corresponds to the decoding function $\mathcal{Y}^{(-1)}$. The parameters of the code are (N, k_c, d_c). The function $\mathcal{Y}$ maps a

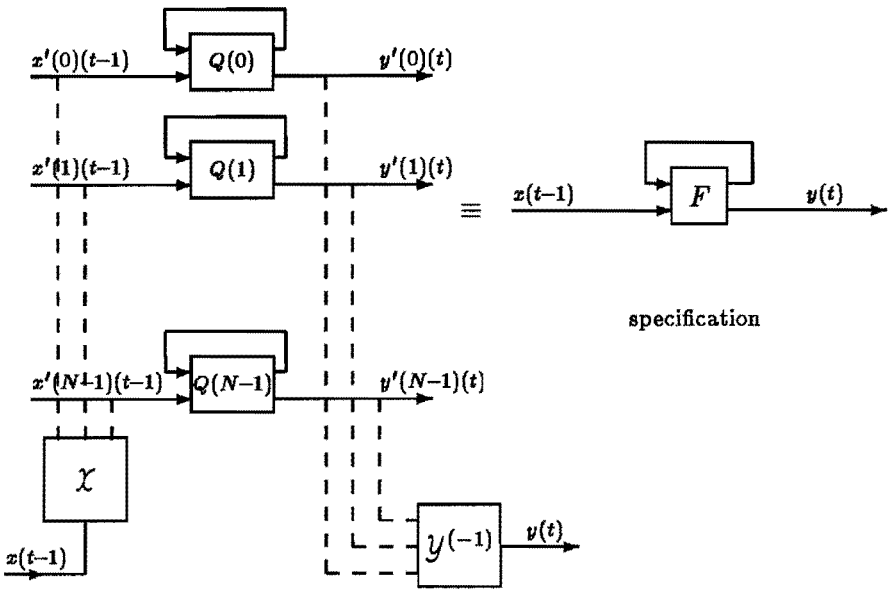


Figure 2.14: *A pictural representation of a* $(\mathcal{X}, \mathcal{Y}, T)$ *fault-tolerant system and its specification*

k_c-tuple of symbols (the data word) onto an N-tuple of symbols (the code word). The function $\mathcal{Y}$ can be divided into N different partial encoding functions $\mathcal{Y}(a)$ with $a \in \mathbf{Ns}$, such that the function $\mathcal{Y}(a)$ applied to the k_c-tuple delivers the a-th symbol of the code word.

In an NMR implementation without state voting like the one depicted in Figure 2.7 on page 64 which is functioning correctly, all outputs $y'(a)(t)$ of the correct modules are identical. So if we apply the partial encoding function $\mathcal{Y}(a)$ to the outputs $y'(a)(t)$, the result will be the a-th symbol of the code word. Clearly, if the decoder function $\mathcal{Y}^{(-1)}$ is applied to this code word and no more than T modules behave maliciously, the original output value $y(t)$ will be retrieved.

Based on this idea, an $(\mathcal{X}, \mathcal{Y}, T)$ design can be built from any specification, cf. Figure 2.16.

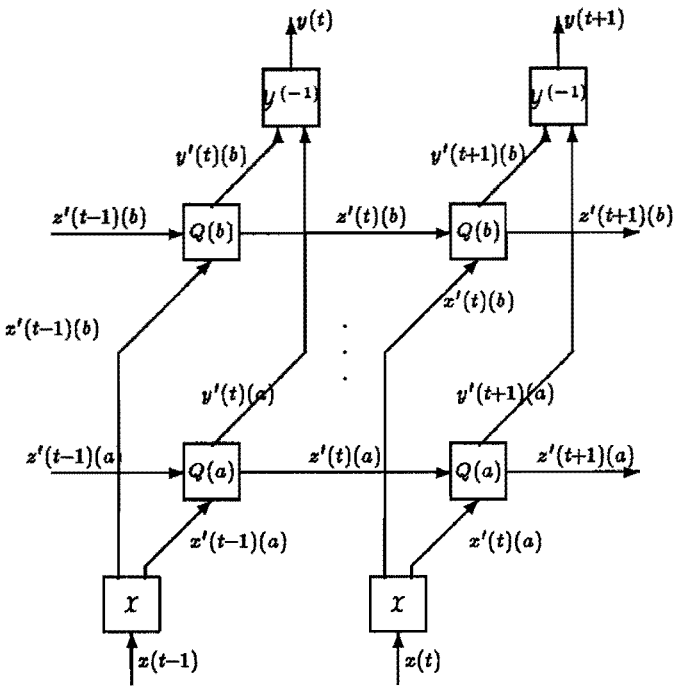


Figure 2.15: *The unfolded representation of a* $(\mathcal{X}, \mathcal{Y}, T)$ *fault-tolerant system consisting of* N *modules. The modules are identified by a variable* a*, with* $a \in \mathbf{Ns}$.

Let the pair of functions F_o, F_s describe the specification. Then

- the distributing $\mathcal{X}$ is a broadcast function ($(N,1)$ repetition code),
- the observing function $\mathcal{Y}^{(-1)}$ is the decoding function of a non-trivial T-error-correcting code. The partial encoding functions corresponding to this code are $\mathcal{Y}(a)$ and
- the behaviour of the N modules is defined by the pair of functions $\mathcal{Y}(a) \circ F_o$, F_s, with $a \in \mathbf{Ns}$.

So the mapping $\mathcal{S}^{(-1)}$ of the N-tuple of states of the implementation to the state of the specification is in this case a majority vote function.

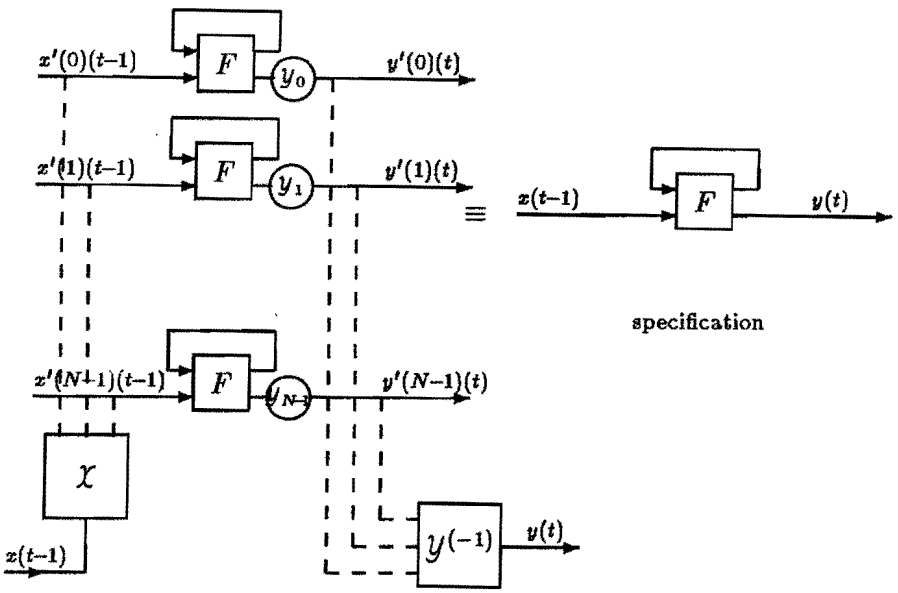


Figure 2.16: *A pictural representation of a $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant system and its specification. The distributing function is a simple broadcast function, the observing function is a non-trivial error-correcting code. A module consists of the concatenation of a copy of the specification and a partial encoder function*

A system like this has the same reliability properties as an NMR system without voting, of which the number of modules is $2T+1$

If $\mathcal{X}$ and $\mathcal{Y}^{(-1)}$ both are the encoding and decoding function of possibly different non-trivial error-correcting codes, other implementations than the one mentioned above, at first glance are less obvious. In fact the only solutions which are known are those in which the specification, thus the comparable non-fault-tolerant system, is extremely simple. Examples are:

- a delay line
- a fixed permutation function, and

- a transmission system.

For instance in a $\mathcal{X}, \mathcal{Y}, T$ fault-tolerant design of a system which delays a stream of words, the functions $\mathcal{X}$ and $\mathcal{Y}$ refer to the same T-error-correcting code with parameters (N, k_c, d_c) and symbol size b_c. The $k_c.b_c$-bit words at the input are first encoded by the distributing function $\mathcal{X}$ into N symbols of b_c bits. Thereafter each of them is delayed in a different module. The observing function $\mathcal{Y}^{(-1)}$ is able to mask the influence of T maliciously behaving modules. Notice that this system belongs to the class of systems with a limited history. The mapping $\mathcal{S}^{(-1)}$ of the N-tuple of states of the implementation to the state of the specification is constructed in this case from three decoder functions which each equal the observing function $\mathcal{Y}^{(-1)}$. The unfolded representation of this design is shown in Figure 2.17. Obviously the reliability properties of this implementation are the same as those of an NMR implementation without state voting, of which the number of modules is $2T + 1$ and which is based in a sequential system with limited history.

The other examples mentioned can be designed in a similar way.

Another interesting design example is as follows:
A large fault-tolerant storage system consists of N disc units, each having their own controller. The distributing function $\mathcal{X}$ is a broadcast function and the observing function $\mathcal{Y}^{(-1)}$ is the decoder function of a (N, k_c, d_c) error-correcting code, which is capable of correcting T random errors, thus $T = \lfloor \frac{d_c - 1}{2} \rfloor$. A message m, which must be stored is broadcast together with the address and a WRITE command to all disc units. In each unit i the controller partially encodes the message m, and stores the result on disc, i.e. the value $\mathcal{Y}(i)(m)$ is stored. $\mathcal{Y}(i)$ is the partial encoder function which delivers the i-th symbol of the code word.
Retrieving information is done by broadcasting the address to all disc units together with a READ command and from the data returned by the units the observer can calculate the message m by applying the decoder function $\mathcal{Y}^{(-1)}$ on the data received. The calculated message is correct provided that no more than T units are malfunctioning.

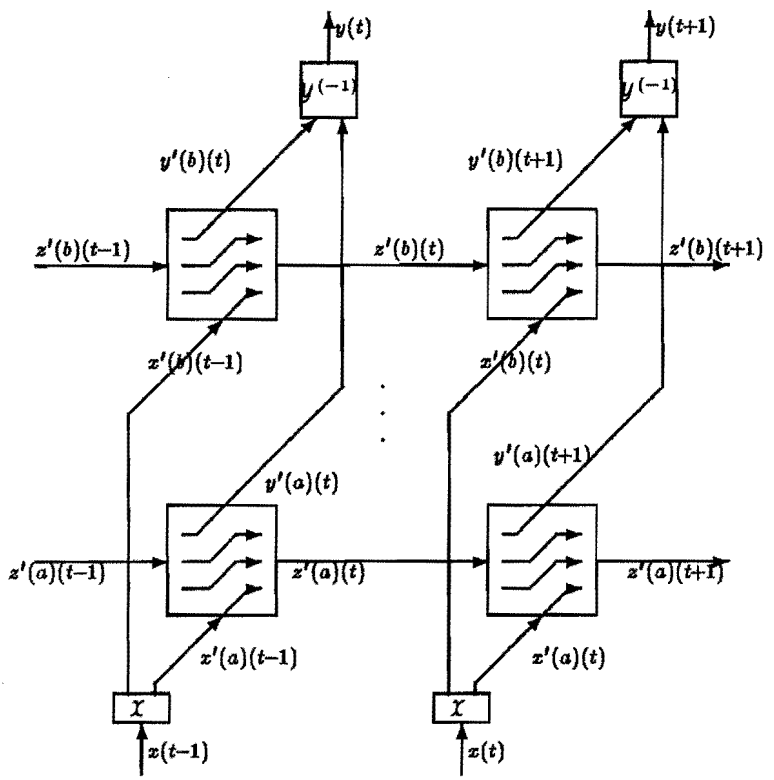


Figure 2.17: *The unfolded representation* $(\mathcal{X}, \mathcal{Y}, T)$ *implementation of a delay line which delays the input* x *over 3 time units. In this case* $\mathcal{X}$ *and* $\mathcal{Y}$ *are the encoder and decoder function of the same error-correcting code respectively.*

2.5.4 $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant systems based on authentication

The preceding description of a $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant system might suggest that the observing function only can be the decoder function of an error-correcting code of which the Hamming distance d_c is related to T by $d_c \geq 2T+1$. This however is not required. Suppose the messages which are broadcast to the modules are authenticated. This means that they are signed and encrypted in such a way that the module can perform a function on it but cannot influence the signature and the encription. The result of a correct function application thus is again signed and encrypted. However, if the module is functioning improperly, the result always needs to be an incorrectly authenticated message. In practice this would mean that (almost) any malfunction of a module can be detected. In that case the minimum number of modules required in the fault-tolerant system is only $T+1$. This type of system is quite similar to those systems which in the literature are called dynamic redundant systems.

The following example is a proper $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant system based on authentication, cf. Figure 2.18, 2.19, and 2.20.
Let the function $\mathcal{X}$ in the previous example (the storage system consisting of N disc units) be as follows:
The message m is encoded by means of a (N,k_c) Maximum Distance Separable code C which is defined over a symbol set. The Hamming distance of such a code is $N-k_c+1$ and the code is able to correct $N-k_c$ symbol erasures. The latter are symbol errors of which the location is known. After encoding we have N symbols, each consisting of a number of bits which is $1/k_c$ times the number of bits of the original message m. Each of these symbols is concatenated with the address (the signature) at which the message has to be stored and thereafter encrypted. Finally these symbols are sent to the corresponding disc units together with the (non-encripted) address and the WRITE command. When a message is read from the storage system, first all received authenticated symbols are decrypted by means of the function *Decript*. The result will be N addresses at which the symbols should have been stored and N symbols of the encoded message. Whatever might have gone wrong during storing and retrieving of the symbol, either the symbol has been mutilated, which results in an incorrect encryption, or a correct encrypted symbol is returned from a wrong address. Both are de-

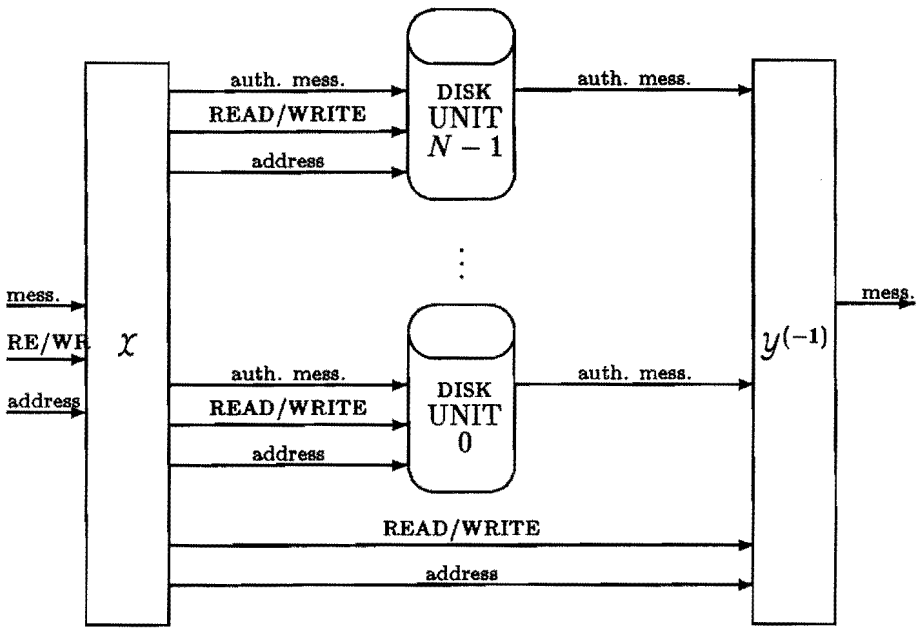


Figure 2.18: *A $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant system based on authentication*

tected (almost) with certainty. Notice that the address was combined with the message before encryption. Provided $N-k_c$ or less of the retrieved symbols are mutilated, the original message can be calculated from the correct symbols and the pointers (erasure flags) to the faulty symbols. Notice that in this case the error-correcting code $\mathcal{C}$ is used for erasure decoding in stead of for decoding random errors.

2.5.5 $(\mathcal{X},\mathcal{Y},\mathcal{Z},T)$ Fault-tolerant systems

In the same way as the $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant systems are obtained from the NMR systems without state voting, the $(\mathcal{X},\mathcal{Y},\mathcal{Z},T)$ fault-tolerant designs are a generalization from the NMR systems with state voting, which is obtained by replacing the state voter by the decoder function $\mathcal{Z}^{(-1)}$ of a T'-error-correcting code, with $T' \geq T$.

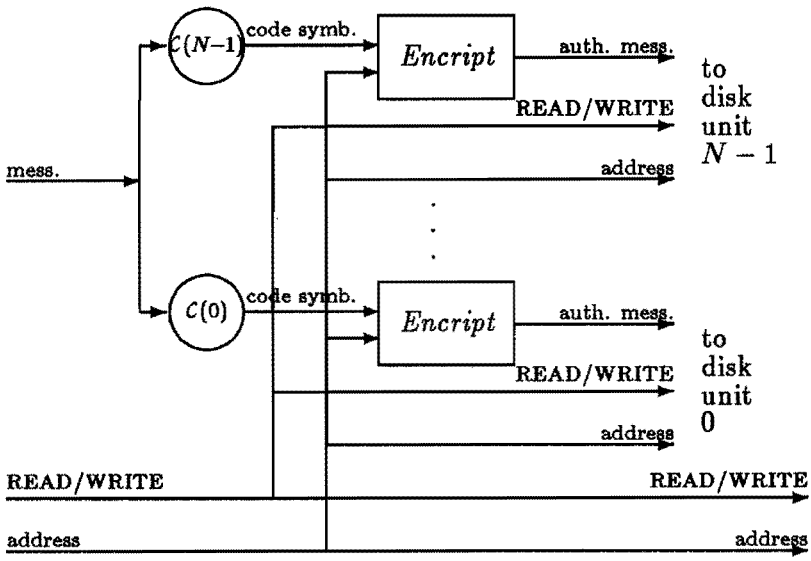


Figure 2.19: *The design of the function $\mathcal{X}$ of the system shown in Figure 2.18*

Recall that the distributing function $\mathcal{X}$ and the observing function $\mathcal{Y}^{(-1)}$ used in the definition of the $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant systems are not a part of the fault-tolerant design but characterize the system and are needed to relate the behaviour which is expressed by the fault-tolerant design to the specification. The N-tuple of states of the modules $(\mathcal{X},\mathcal{Y},T)$ fault-tolerant systems were related to the state expressed by the specification by the function $\mathcal{S}^{(-1)}$, which is the decoder function of an error-correcting code.

In contrast to this, in an $(\mathcal{X},\mathcal{Y},\mathcal{Z},T)$ fault-tolerant system the function $\mathcal{Z}^{(-1)}$ is used to relate the N-tuple of states in the fault-tolerant design to the state expressed by the specification. The function $\mathcal{Z}^{(-1)}$ thus is used in two ways, firstly as a piece of hardware available in each module in order to re-initialize the module each time instance and secondly for comparing the N-tuple of states of the implementation with the state of the specification.

So $(\mathcal{X},\mathcal{Y},\mathcal{Z},T)$ fault-tolerant systems are built from N fully interconnected

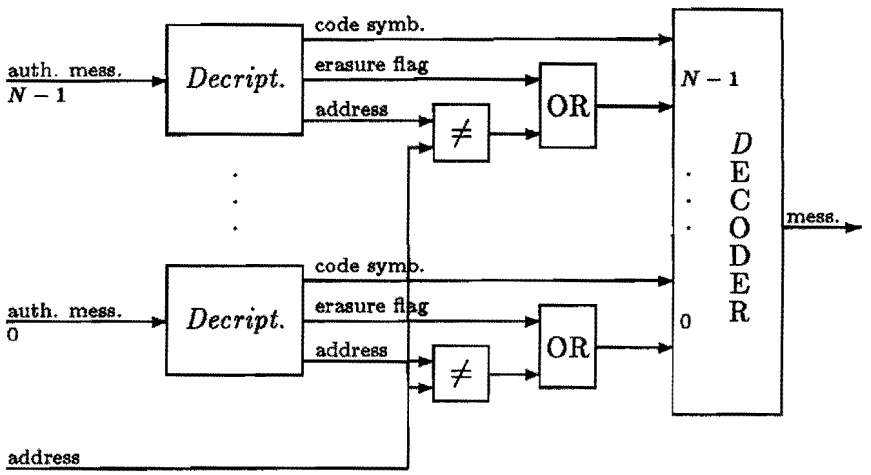


Figure 2.20: *The design of the function* $y^{(-1)}$ *of Figure 2.18*

modules as is depicted in Figure 2.21 and are able to tolerate T faulty modules. The modules in the design are identified by the elements of a set **Ns** and the behaviour of the modules is specified by the functions $z^{(-1)}$ and $Q(a)$ with $a \in$ **Ns**. The distributing function x and the observing function $y^{(-1)}$ are not a part of the fault-tolerant design but characterize the system and are needed to relate the behaviour which is expressed by the fault-tolerant design with the specification. Thus:

Definition 2.2 *A system belongs to the class of* (x, y, z, T) *fault-tolerant systems if the behaviour of the fault-tolerant implementation together with the functions* x *and* $y^{(-1)}$*, in the presence of* T *or less faulty modules, is equivalent to the behaviour expressed by the specification.*

The unfolded representation of an (x, y, z, T) fault-tolerant system is shown in Figure 2.22. From the definition of the class of (x, y, z, T) fault-tolerant systems, the definition of NMR systems with state voting and a comparison of the Figures 2.21 and 2.22 with the Figures 2.9 and 2.10, it immediately follows that the reliability properties of (x, y, z, T) fault-tolerant implemen-

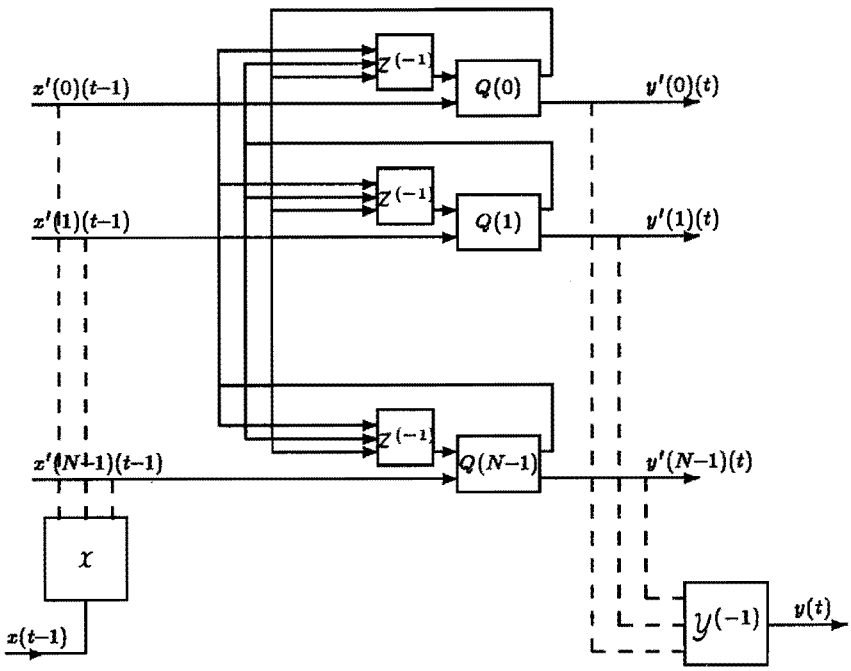


Figure 2.21: *A pictural representation of a* $(\mathcal{X}, \mathcal{Y}, Z, T)$ *fault-tolerant system with state voting*

tations are identical to those of NMR implementations with state voting, of which the number of modules is $2T + 1$.
Obviously the class of NMR systems with state voting is a subclass of the class of $(\mathcal{X}, \mathcal{Y}, Z, T)$ systems.

2.5.6 Examples of $(\mathcal{X}, \mathcal{Y}, Z, T)$ fault-tolerant systems

In principle for any system which is specified according to the Moore model and for any T an $(\mathcal{X}, \mathcal{Y}, Z, T)$ design can be made straightforwardly. Such a design is depicted in Figure 2.23.

If we let the specification of the system be described by the pair of functions

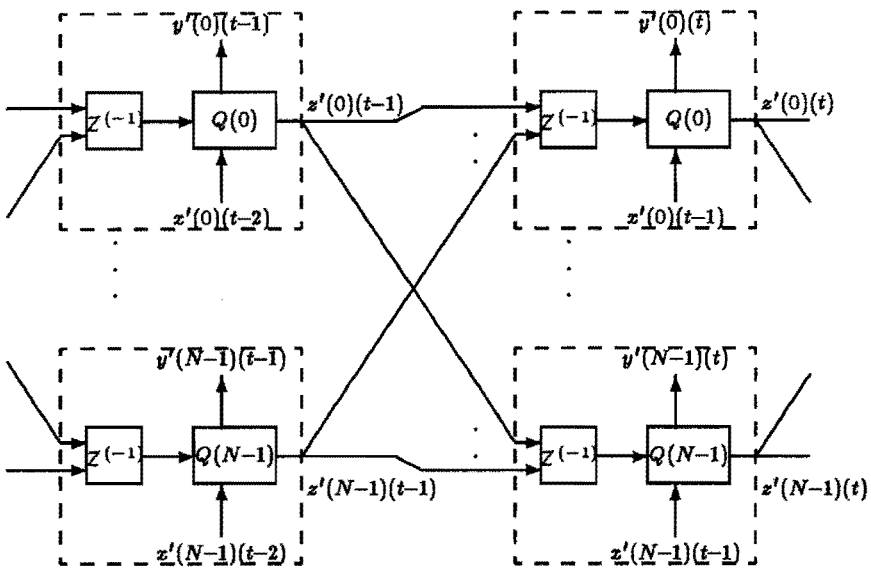


Figure 2.22: *The unfolded representation of a* $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ *fault-tolerant design. The distributing function* $\mathcal{X}$ *and the observing function* $\mathcal{Y}^{(-1)}$ *are omitted*

F_o, F_s, then the design is constructed as follows:

- the distributing function $\mathcal{X}$ is a broadcast function $((N,1)$ repetition code),
- the observing function $\mathcal{Y}^{(-1)}$ is the decoding function of a non-trivial T'-error-correcting code, with $T' \geq T$. The partial encoding functions corresponding to this code are denoted $\mathcal{Y}(a)$, with $a \in \mathbf{Ns}$, and
- the state decoding function $\mathcal{Z}^{(-1)}$ is the decoding function of a non-trivial T''-error-correcting code, with $T'' \geq T$. The partial encoding functions corresponding to this code are denoted by $\mathcal{Z}(a)$, and
- the behaviour of the N modules is defined by the pair of functions $\mathcal{Y}(a) \circ F_o$, $\mathcal{Z}(a) \circ F_s$, with $a \in \mathbf{Ns}$.

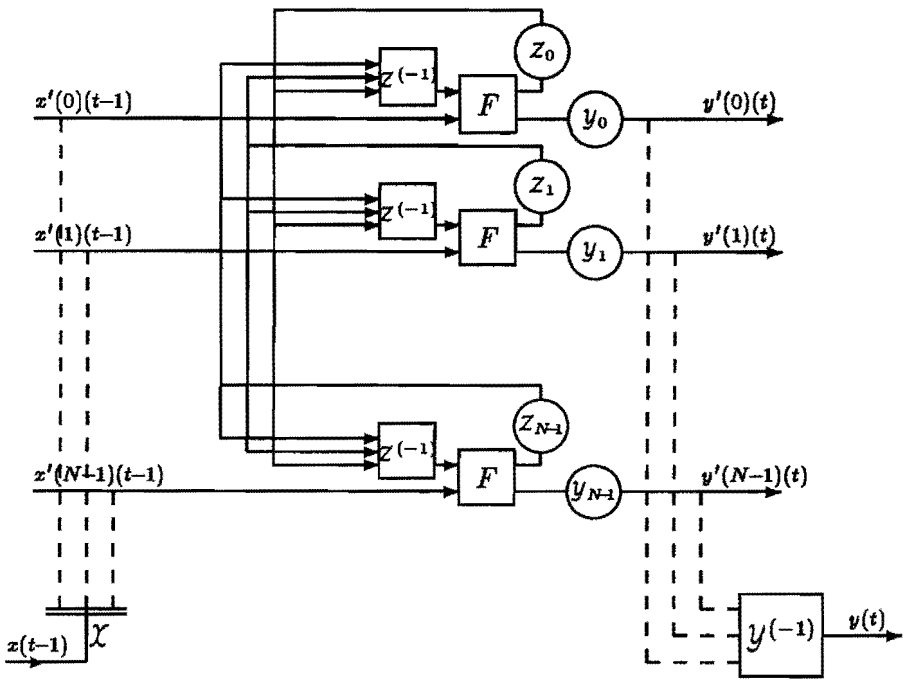


Figure 2.23: *A pictural representation of a straightforwardly obtained* $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ *fault-tolerant design*

The choice of the code can only be made on criteria outside the scope of this discussion and in general will depend on the specification. For this reason we will only make a few remarks.

Further on in this thesis we will show that the Input Problem can only be solved if the number of modules is at least $3T + 1$, while an NMR implementation with or without state voting only requires $2T + 1$ modules. For economical reasons it better to have all modules as identical as possible. Therefore the class of $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ design offers the possibility to match the number of modules with the requirement which stems from the Input problem.

In Figure 2.23 the data offered to the next column in the systolic array is encoded by means of the partial encoding functions $Z(a)$, which are derived from the encoding function Z. Clearly this could be any T-error-correcting code. Each module must produce one symbol of the encoded state. Suppose we choose for this code a (N, k_c) maximum distance separable code. In that case the system is capable of correcting the influence of $T = (N - k_c)/2$ randomly failing modules. The minimum value of k_c is of course 1. The amount of hardware which is required for the implementation of the functions is proportional to N. Thus for fixed T, the amount of hardware required tends to increases proportionally with k_c, depending on the application. However the amount of data to be transmitted (N modules sending symbols to N modules) is proportional to N^2/k_c, which is minimal for $k_c = 2T$. Thus in that sense the $(4,2)$-concept is optimal. If the cost function is determined by the amount of data produced by the modules, i.e. the cost function is proportional to $\frac{N}{k_c} = \frac{2T}{k_c} + 1$, then the cost of the system decreases with increasing k_c.

Clearly an $(\mathcal{X}, \mathcal{Y}, Z, T)$ implementation of a system which has a large state space is for practical reasons not possible. Therefore another solution will be presented, which in fact is a mixture of an $(\mathcal{X}, \mathcal{Y}, T)$ and an $(\mathcal{X}, \mathcal{Y}, Z, T)$ design.

In systems with a large state space in most cases something like a memory can be recognized. Typically in a von Neumann computer, the state of the system is determined by the content of the memory, the registers and the counters. The result of a function executed at a particular time instance in a von Neumann computer, only depends on a part of the state information and possibly on the input. To be more precise, only the content of a few registers, the counters and the content of the selected memory location(s) can be used as input for the function. Hence only that part of the state information has to be available in the form of plain data, the rest may remain encoded. This leads to a system in which the largest part of the state information is transferred to the next time instance of the same row in the systolic array in the form of encoded data without interference with the other rows and without being affected by the function. These data typically are the encoded memory data. A small part of the encoded data may be affected by the function, i.e. due to write operations on the memory. A part of the state information is transferred in the form of plain data without interference with the other rows. This is typically the register and counter data, when it is not

affected by the function. Some of this plain data may be (partly) changed by the function application. Only a small part of the encoded state information is broadcast to the other rows. The decoder function is applied to this data in order to obtain a common result among the correctly functioning modules.

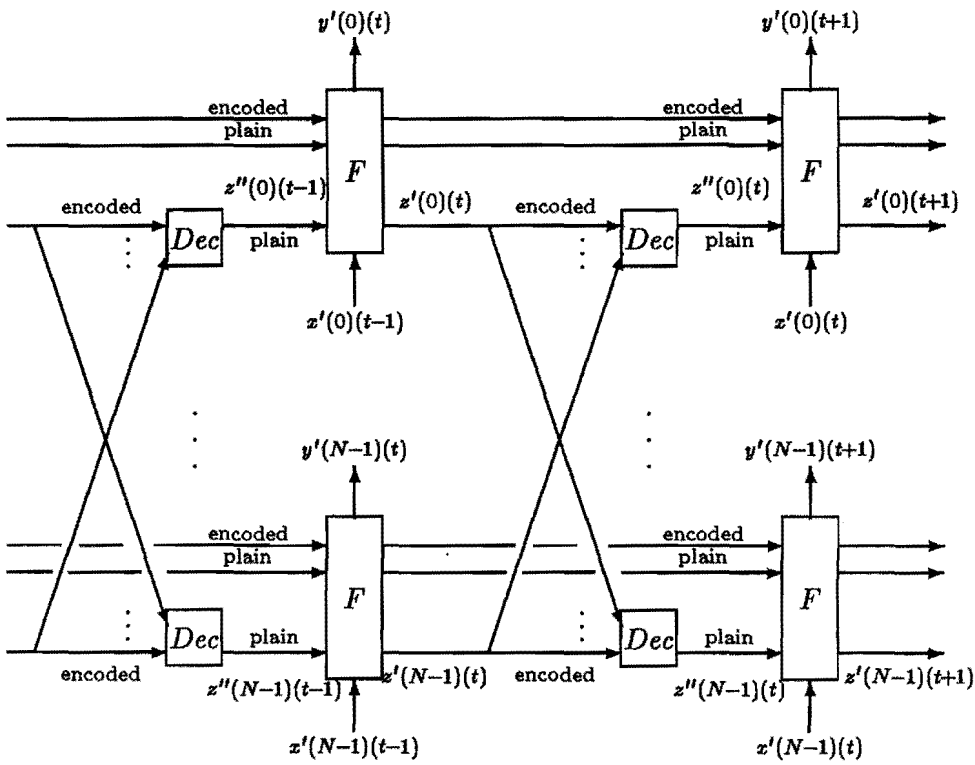


Figure 2.24: *The unfolded representation of a von Neumann machine on which the (N, K)-concept applied*

The basic principle of such a system is shown in Figure 2.24. A typical example of a system like this is the (N, K)-concept as it has been described in Section 2.2. In order to elucidate the relation between the typical von Neumann machine based description in Section 2.2 and the description in this section by means of unfolding the time, in Figure 2.25 one cell of the systolic array in Figure 2.24 is shown, which corresponds to the architecture

in Figure 2.2 on page 45. The cell is divided into an ALU part and an address decoder part. In each part the functional dependencies between input and output of both the ALU and the address decoder are shown.

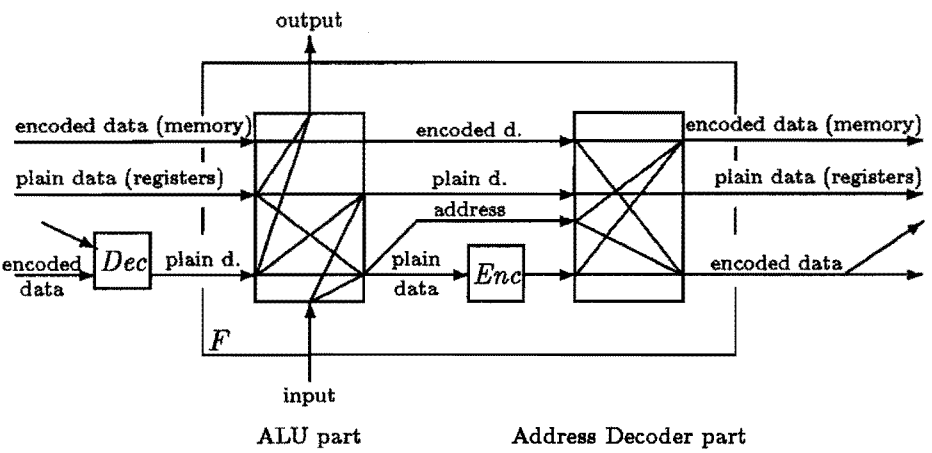


Figure 2.25: *One cell of the systolic array in Figure 2.24 in detail*

In $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant systems at each time instance the state decoder $\mathcal{Z}^{(-1)}$ is applied in each module on the entire combined state of all modules. In $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant systems the state of a module is not affected by the other modules and can only be affected by means of the input of the system.

In the (N, K)-concept each time instance the state decoder $\mathcal{Z}^{(-1)}$ is in each module is only applied on a small part of the combined state of all modules. Hence this part operates according to a $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant system. The remaining state information, which in Figure 2.24 is indicated by the uppermost two inputs of the function F, is not affected by the other modules. Hence this part operates according to a $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant system.

Each part of the state information is accessible by the state decoders and of course each part of the state information can be changed. So the system itself determines which part operates according to an $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant system and which part operates according to an $(\mathcal{X}, \mathcal{Y}, T)$ fault-tolerant system.

From the preceding it follows that in a system such as the one depicted in Figure 2.24, during a single time instance only a part of the encoded data, and in general also only a part of the plain data, can be replaced by a common value. So re-initialization of the system will take many time instances. We will elaborate on this problem in detail when describing the $(4,2)$-concept in the next section.

The previous discussion shows that, starting from a specification, fault-tolerant systems can be designed and analyzed in a top-down process. The different concepts of fault-tolerance, $(\mathcal{X},\mathcal{Y},T)$ fault-tolerance and $(\mathcal{X},\mathcal{Y},\mathcal{Z},T)$ fault-tolerance may be combined arbitrarily. The functions $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ may be any T-error-correcting code or a function which provides error-correction on the basis of authentication. If a fail-stop or fail-safe system is required, error-detecting codes may be used.
The previous discussion also shows that unfolding of time into space is a simple method which can be used for a better understanding of the behaviour of the fault-tolerant design. Playing with pictural representations of unfolded systems leads to many different fault-tolerant architectures.

The time axis of the systolic arrays in the previous examples also might be interpreted as an axis in space. The fault-tolerance properties are independent of the interpretation of these axes, so fault-tolerant systolic arrays can be designed according to the same philosophy.
Another freedom in the design process stems from the fact that the time between two successive time instances, i.e. one time unit, might be composed from a number of sub-time units. Hence the functional boxes in the previous examples may also be interpreted as finite state machines.

Each of the modules in the systolic array may be again a fault-tolerant system. This idea for instance could be applied to construct extremely reliable communication networks.

2.6 The $(4,2)$-concept

2.6.1 System description

The $(4,2)$-concept consists of four modules, Figure 2.26. The processor part is quadrupled as compared with a non-redundant computer, whilst the

memory consists of four parts, each with a word length (symbol size) of half a data word. So the memory is only doubled.

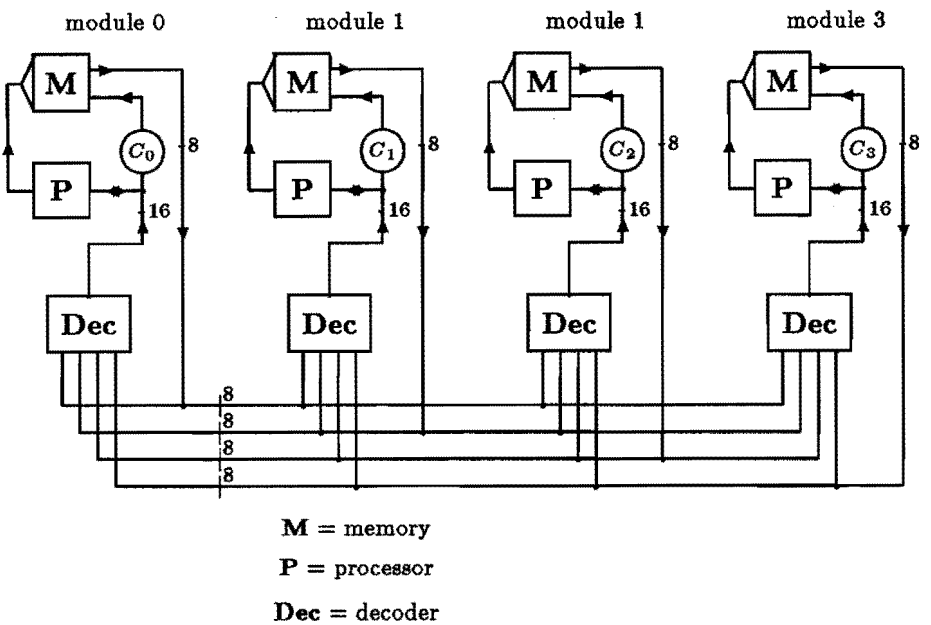


Figure 2.26: *The* (4,2)*-concept*

The information stored in the memories of the four modules is protected by means of a symbol- and bit-error-correcting code. Because the data word length is 16 bits, the code consists of four 8-bit symbols, two of which are to be regarded as information symbols while the other two are check symbols. This code is built up from two interleaved codes, both consisting of four symbols of four bits. The code can correct all possible single symbol-errors and all possible double bit-errors even if these bit-errors are in different modules.
The four processors, one in each module, contain identical information and run synchronously.

2.6.2 Data transfer between processor and memory

Data transfer between processor and memory proceeds as follows. When information is sent from the processor to the memory (i.e. a WRITE opera-

tion), the 16-bit data word is encoded into 8 bits in each module and written into the memory of that module. The encoding rules for the four modules are different in such a way that the four 8-bit symbols together form a word of the afore-mentioned symbol- and bit-error-correcting code. Notice that the hardware implementation of the modules differs only in the encoders.

When information has to be transferred from the memory to the processor (i.e. a READ operation), each module will receive not only the 8-bit symbol which is stored in its own module but also the other three symbols of the code word stored in the other modules.
So each module receives the complete code word of four 8-bit symbols. If any one of these symbols should be wrong or two bits possibly in different modules should be wrong, it can be corrected by the decoders available in each module. The only interconnections between the modules are the 8-bit symbols which are interchanged between the modules. This means that, whatever fault occurs in a module, it can affect only the 8-bit symbol that is sent to the other modules. These modules, however, can correct the fault. In other words, a hardware fault, as long as it is limited to one module or affects only two bits in different modules, does not affect the functioning of the system.

Both the (4,2)-concept and a triplicated system tolerate one failing fault-isolation area, so their reliability improvement must be of the same order of magnitude.

2.6.3 Applicable symbol-error-correcting codes for the (4,2)-concept

Basically, in the (4,2)-concept, any single symbol-error-correcting code of the maximum distance separable (MDS) type can be applied, e.g. a Reed-Solomon code [MacW 78]. The minimum symbol size b of an MDS code which is defined over symbols from the binary extension field is determined by $b \geq \log_2(N+1)$, in which N is the word length of the code, [MacW 78]. Hence it follows that a four-symbol MDS code requires a minimum symbol size of two bits. Codes with 8-bit symbols can simply be derived from it by interleaving four of these codes. This offers the advantage that each decoder can be split up into four decoders, each acting on 2-bit symbols. In non-interleaved codes based on larger symbol sizes a lot of redundancy is left unused. However in some codes this redundancy can be used for additional

bit-error-correcting capabilities. Most 16-bit microprocessors are able to read and write single 8-bit bytes, therefore a code word must not contain more than 8 bit data. Thus, because the data is stored in two symbols, the symbol size is restricted to four bits.
An exhaustive computer search led to an optimal (4,2) symbol-error-correcting code with 4-bit symbols, the mathematical description of which is given in Section 2.7.
The decoder of this code can operate in different modes. Each mode results in a different decoding strategy, depending on preliminary knowledge of the fault behaviour of the system.

The code has the following characteristic properties. In the "random mode" it corrects:

- any single symbol-error and
- all double bit-errors, even if they are not located in the same symbol.

(double bit-errors and single symbol-errors cannot be corrected simultaneously)
In the "erasure mode" it corrects:

- any single bit-error in the presence of a symbol erasure (the latter is a symbol-error whose location is already known by the decoder).

Notice that there are four erasure modes, each pointing to a different suspicious module.
Next to the random mode and the four erasure modes, there are six single modes. Due to the MDS property of the code the data word can be derived from any two correct symbols of the code word. So the system is able to run faultfree on only two modules, provided these modules are working correctly. If the system runs on two modules, the symbols in the code word originating from the other two modules are considered by the decoders as erasures. Clearly in this mode no additional correction or detection capacity is left.

Often the failure rate of the memory is predominant. But when the memory is bit-sliced, a single failing memory chip only influences one bit of the code word and thus produces only bit-errors. Most of them are transient. So the profitability of the additional bit-error-correcting properties of the code is obvious.

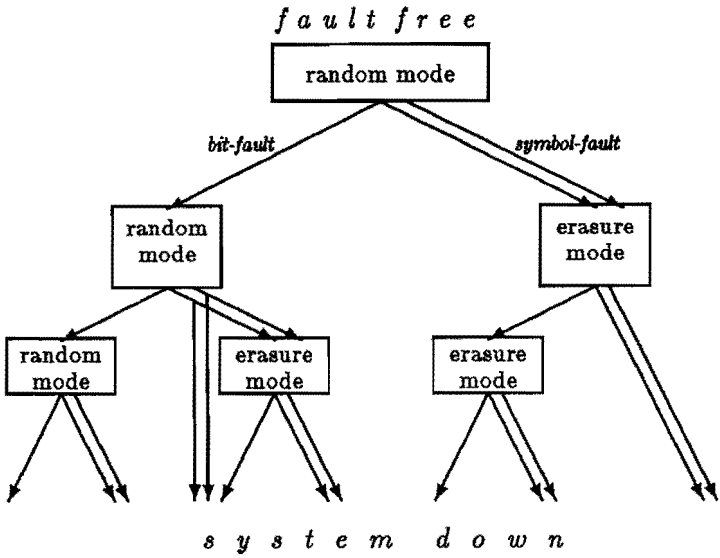


Figure 2.27: *The state transfers of the decoder*

Under fault-free conditions, the system operates in the random mode, in which all the single symbol-errors and all double bit-errors are corrected. The system remains in the random mode when a bit-error occurs, but as soon as the decoders detect and correct a symbol-error that is not a bit-error, they automatically switch to the erasure mode indicating the failing module, (cf. Figure 2.27). Hence the code word which is read immediately after the code word containing a symbol-error is already decoded in the erasure mode.

Suppose a repair time sets in when a permanent bit fault or a symbol fault is corrected. Then the system will only go down if two additional bit faults or a symbol fault arise during repair time. However, when the fault initiating the repair time is a bit fault and the second fault is a symbol fault, there is a good chance that the first fault will not be effective at that instant and the system will survive because it immediately switches to the erasure mode.

From the foregoing it is clear that the memory failure rate hardly influences the mean time between down when the (4,2)-concept is provided with

symbol- and bit-error-correcting code.

Similar properties could have been obtained by applying a Reed-Solomon code over 4-bit symbols and adding a parity bit to each symbol. The decoder complexity would have been the same but the memory hardware would have been 2.5 fold instead of 2 fold compared to a single non-redundant system.

2.7 Symbol- and bit-error-correcting codes

In this section we will describe the symbol- and bit-error-correcting code for the (4,2)-concept fault-tolerant computer which was found by means of an exhaustive computer search. The properties of this code however can be proved mathematically.
The code is defined as the null space of its parity check matrix H:

$$H = \begin{pmatrix} \alpha^7 & \alpha^{11} & \alpha^0 & \varphi \\ \alpha^{11} & \alpha^7 & \varphi & \alpha^0 \end{pmatrix} \tag{2.14}$$

where α is a root of the primitive polynomial $x^4 + x + 1$ which generates $GF(2^4)$ and φ is the zero element of $GF(2^4)$. Thus this block code is a two-dimensional subspace of the four-dimensional vector space $(GF(2^4))^4$. The symbols of $GF(2^4)$ again form a four dimensional vector space over $GF(2)$. The basis of this vector space is constituted by α^3, α^2, α^1, and α^0. These symbols correspond to the basis vectors (1000), (0100), (0010), and (0001) respectively.
A generator matrix G for this code is:

$$G = \begin{pmatrix} \alpha^0 & \varphi \\ \varphi & \alpha^0 \\ \alpha^7 & \alpha^{11} \\ \alpha^{11} & \alpha^7 \end{pmatrix} \tag{2.15}$$

A number of properties of the code will be derived, using the following definitions:

- The symbol weight w_s of a code word is the number of non-zero symbols in that code word.
- The minimum symbol weight w_{sc} of the code is the minimum symbol weight taken over all non-zero code words of the code (which equals the symbol distance of the code).

- The bit weight w_{bs} of a symbol is the number of non-zero bits in that symbol.
- The bit weight w_b of a code word is the total number of non-zero bits in the code word.

Lemma 2.1 *The minimum symbol weight w_{sc} of the code, which is defined by the parity matrix in equation (2.14), is 3.*

Proof
By inspection we see that any 2×2 sub-matrix of H is non-singular and the generator matrix shows the existence of a code word with symbol weight 3. □

Lemma 2.2 *No code word exists with a symbol weight 4, where the bit weight of all symbols is 1.*

Proof
Suppose the existence of a code word with symbol weight $w_s = 4$ where the bit weight of each symbol is $w_{bs} = 1$. Thus all symbols of the code word are taken from the set $\{\alpha^3, \alpha^2, \alpha^1, \alpha^0\}$. Then from the generator matrix it follows that the following relations must hold:

$$\alpha^{7+i} + \alpha^{11+k} \in \{\alpha^0, \alpha^1, \alpha^2, \alpha^3\} \tag{2.16}$$

and

$$\alpha^{11+i} + \alpha^{7+k} \in \{\alpha^0, \alpha^1, \alpha^2, \alpha^3\} \tag{2.17}$$

in which i and $k \in \{0, 1, 2, 3\}$.

Calculating that part of the addition table of $GF(2^4)$, which represents

$$\alpha^{7+i} + \alpha^{11+k}$$

as a power of α with i and $k \in \{0, 1, 2, 3\}$, we obtain the following table in which only the exponents of α are shown.

$\alpha^{7+i} + \alpha^{11+k}$		k			
		0	1	2	3
	0	8	2	5	1
i	1	7	9	3	6
	2	2	8	10	4
	3	14	3	9	11

From this table we deduce that the preceding relations (2.16) and (2.17) never hold simultaneously.

□

Lemma 2.3 *No code word exists with symbol weight 3 where two or three symbols have bit weight 1.*

Proof
From the generator matrix it follows that all code words with symbol weight $w_s = 3$ are described by:

$$(\varphi\ ,\ \alpha^i\ ,\ \alpha^{11+i}\ ,\ \alpha^{7+i})\ ,\ (\alpha^i\ ,\ \varphi\ ,\ \alpha^{7+i}\ ,\ \alpha^{11+i})\ ,$$
$$(\alpha^{4+i}\ ,\ \alpha^i\ ,\ \varphi\ ,\ \alpha^{9+i})\ \text{ and }\ (\alpha^i\ ,\ \alpha^{4+i}\ ,\ \alpha^{9+i}\ ,\ \varphi) \tag{2.18}$$

with $i \in \{0, 1, \cdots, 14\}$.

The symbols of bit weight 1 are in the set $\{\alpha^0, \alpha^1, \alpha^2, \alpha^3\}$.
Inspection of the above code words shows that no two or three symbols of a code word can be in this set simultaneously.

□

Theorem 2.4 *The code defined by the parity check matrix H given above has the property that:*

- *Two different single symbol-errors never result in the same syndrome.*
- *Two different double bit-errors never result in the same syndrome.*
- *The set of all single symbol-errors and the set of all double bit-errors are disjoint.*

Proof
The first property follows immediately from lemma (2.1) and the second property from lemma (2.2) and (2.3).
The last property can be derived from lemma (2.1) and (2.3). When a double bit-error has the same syndrome as a single symbol-error, the bit-by-bit sum of these error vectors must be a code word. So a code word should exist with symbol weight 3 of which two of these symbols should have bit weight 1. Or this code word should have symbol weight 2. However, both cases are excluded by the lemmas (2.1) and (2.3).

□

Theorem 2.5 *The code defined by the above parity check matrix H is capable of correcting single bit-errors in the presence of an erased symbol.*

Proof
Remember that the location of an erased symbol but not the error value is known. Again the bit-by-bit sum of any two correctable errors should not be a code word. Thus the sum of two symbol-erasures at the same location and two random bit-errors must not be a code word. This would imply the existence of a code word of symbol weight 2 or a code word of symbol weight 3 of which two symbols have bit weight 1. This again is excluded by lemma (2.1) and (2.3). □

It is very likely that symbol- and bit-error-correcting codes can be found, for any value of N and K, provided the symbol size is sufficiently large.
As an example a $(3,1)$ symbol- and bit-error-correcting code can be obtained from the $(4,2)$ code described in this paper by just shortening the $(4,2)$ code by one symbol. The parity check matrix of this code follows from the parity check matrix of the $(4,2)$ code by deleting one column. However this code certainly is not the best code that can be found in the $(3,1)$ case. In [Gils 86], [Gils 87], [Gils 88] and,[Boly-88] a large class of combined symbol- and bit-error-correcting codes are described.

2.8 Decoding symbol- and bit-error-correcting codes

The propagation delay of the decoders must be as small as possible because this delay has to be added to the memory access time and so influences the performance of the system. Therefore the decoders must be implemented as a combinatorial network.
The complexity of the decoders is such that they should preferably be implemented in LSI.

An easy-to-implement decoder which is in principle applicable for any (N,K) symbol- and bit-error-correcting code can be based on deriving the syndrome from the code word received, thereafter deriving the error pattern from this syndrome and then subtracting this error pattern from the code word received.

The syndrome is found by multiplying the code word received by the parity check matrix which can be implemented by means of
EXCLUSIVE-OR gates. Because there is a one-to-one relation between syndromes and the error patterns, the mapping from the syndrome into the error pattern can be implemented with a ROM. This decoding principle is depicted in Figure 2.28.
A decoder like this is however not optimal in terms of propagation delay and hardware. Moreover the size of the ROM grows exponentially with the number of check bits, i.e. $(N-K)L$. In this context L is the symbol size of the code in bits.

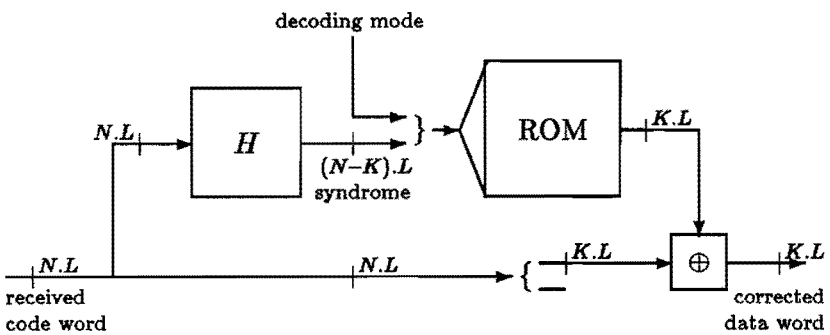


Figure 2.28: *A simple and fast read-only memory based decoder*

In the remainder of this section a new decoder principle will be presented which is very well suited for decoding combined symbol- and bit-error-correcting codes. This principle will be explained with the help of the decoder for the $(4,2)$ code which has been defined in the previous section. The design is based on LSI implementation and optimized for minimum propagation delay. The decoding principle can be generalized to other values of N and K, as for instance has been done in [Gils 86] for $N=3$ and $K=1$.

The data words in the $(4,2)$ code can always be recovered from any two fault-free symbols in the code word. This follows from the MDS properties, [MacW 78]. The correction capabilities of the code show that there are never more than two faulty symbols in the code word. So for correction it is only necessary to locate two fault-free symbols and to derive the data word from

these symbols.
Therefore the decoder consists of two parallel working parts, i.e.:

- A circuit that calculates 6 versions of the data word from the 6 pairs of symbols of the received word.
- A circuit that calculates first the syndrome and then determines from this which symbols are faulty and thus which version(s) of the data word is (are) correct.

The circuitry for deriving the six versions of the data word from the received word is based on the property that each of the six square submatrices of the generator matrix G are nonsingular. Hence each of the six versions of the data word can be derived by taking the reciprocal of the corresponding square submatrix of G and multiplying this with the corresponding two symbols of the word received.

In order to find the error locations (i.e which symbols are erroneous), we start from an expanded parity check matrix Q. In this expanded parity check matrix two rows are added to the original matrix, such that:

- The two rows added are a linear combination of the rows in the original parity check matrix,
- All the 2×2 submatrices of the 4×4 matrix Q are nonsingular.
- Each row of the matrix Q contains an entry equal to φ, such that all columns contain one φ.

Notice that the original parity check H may be multiplied by an arbitrary 2×2 nonsingular matrix without changing the code.
There are many solutions for the matrix Q, for example the matrix Q in:

$$\mathbf{s}^{tr} = \begin{pmatrix} s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} \varphi & \alpha^0 & \alpha^{10} & \alpha^6 \\ \alpha^0 & \varphi & \alpha^6 & \alpha^{10} \\ \alpha^{11} & \alpha^7 & \varphi & \alpha^0 \\ \alpha^7 & \alpha^{11} & \alpha^0 & \varphi \end{pmatrix} \begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = Q.\mathbf{c}^{tr} \qquad (2.19)$$

The syndrome vector s has the following properties:

- If there is no error all symbols of s are φ.

- If there is a single symbol-error, one symbol of **s** equals φ and the other three are unequal to φ. So from the matrix Q it follows immediately that the location of the φ symbol in **s** indicates the erroneous symbol in the word received.
- If there is a double bit-error in which the erroneous bits are located in different symbols all the symbols of **s** are nonzero.

The proof of the first property is trivial, the second property follows from the fact that any two rows of Q define the code, and the last property can be proved as follows:
Suppose one of the symbols of **s** equals φ in the case of a double bit-error of which the erroneous bits are in different symbols. Let the index of this φ in **s** be l. Consider the l-th row of Q. None of the erroneous bits will be in the l-th symbol. Consider an error pattern consisting of the previously defined double bit-error and a symbol-error at location l. This error pattern, after multiplication by Q, will also result in $s_l = \varphi$. The value of the symbol-error can always be chosen so that the error pattern also causes another φ in **s**. But any two rows of Q define the code and the correction properties of the code require that this error pattern results in a non-zero syndrome, which contradicts the zero symbol in **s**.

The conditions on which the symbol locations of a double bit-error can be found from the syndrome vector **s** are as follows.
Let the error vector $\mathbf{e} = (e_3 e_2 e_1 e_0)$ represent a double bit-error on the symbol locations i and j $(i \neq j)$ and let the two non-zero symbols of **e** at the locations i and j be represented by

$$e_i = \alpha^f \quad \text{and} \quad e_j = \alpha^g$$

in which $i, j, f, g \in \{0, 1, 2, 3\}$
Let the k-th row of Q be

$$\mathbf{h}_k = (h_{k,3}, h_{k,2}, h_{k,1}, h_{k,0})$$

Thus

$$s_k = \mathbf{h}_k.\mathbf{c}^{tr} = \mathbf{h}_k.\mathbf{e}^{tr} = (h_{k,i}.\alpha^f + h_{k,j}.\alpha^g)$$

Because $h_{i,i} = \varphi$ and $h_{j,j} = \varphi$ we find

$$s_i = \mathbf{h}_i.\mathbf{e}^{tr} = h_{i,j}.\alpha^g$$

and

$$s_j = \mathbf{h}_j.\mathbf{e}^{tr} = h_{j,i}.\alpha^f$$

Thus s_i is not influenced by an error at location i and s_j is not influenced by an error at location j.

Let

$$h_{i,j} = \alpha^{i_1} \quad \text{and} \quad h_{j,i} = \alpha^{j_1}$$

then

$$s_i = \alpha^{i_1+g} \quad \text{and} \quad s_j = \alpha^{j_1+f}$$

Thus for arbitrary bit-errors at symbol locations i and j

$$s_i \in \{\alpha^{i_1}, \alpha^{i_1+1}, \alpha^{i_1+2}, \alpha^{i_1+3}\}$$

and

$$s_j \in \{\alpha^{j_1}, \alpha^{j_1+1}, \alpha^{j_1+2}, \alpha^{j_1+3}\}$$

Double bit-errors not located in symbol i and j can never cause the syndrome satisfying the previous condition, because that would imply that two double bit-errors result in the same syndrome. (Remember that any two rows of Q are a complete parity check matrix of the code.) Thus these conditions are sufficient for deriving the symbol locations of a double bit-error (which is not a symbol-error).

For determining the symbol location of the bit-error in erasure mode i which points to an erasure symbol-error in module i, only row i of Q has to be considered. The erasure symbol-error does not influence the value of s_i, so the value of s_i uniquely determines the location of the bit-error.

Starting from correctable error patterns, the conditions on which the error locations are found can be summarized as follows:

- **Random mode:**

$$[\ \exists i,j\ :\ i \neq j \wedge s_i = \varphi \wedge s_j = \varphi\ ] \Longleftrightarrow \underline{\text{No Error}}$$

$$\left(s_i = \varphi \wedge [\ \forall j\ :\ s_j \neq \varphi\ ]\right) \Longleftrightarrow \underline{\text{Single Symbol-error at Location } i}$$

$$\left(i \neq j \wedge s_i \in \{\alpha^{i_1}, \alpha^{i_1+1}, \alpha^{i_1+2}, \alpha^{i_3}\} \wedge s_j \in \{\alpha^{j_1}, \alpha^{j_1+1}, \alpha^{j_1+2}, \alpha^{j_3}\}\right)$$

$$\Longleftrightarrow \underline{\text{Double Bit-error Locations } i \text{ and } j}$$

in which $\alpha^{i_1} = h_{i,j}$ and $\alpha^{j_1} = h_{j,i}$.

- **Erasure mode:** erasure at location i

$$s_i = \varphi \Longleftrightarrow \underline{\text{No Bit-error}}$$

$$\left(s_i \in \{\alpha^{i_1}, \alpha^{i_1+1}, \alpha^{i_1+2}, \alpha^{i_3}\}\right) \Longleftrightarrow \underline{\text{Bit-error at Location } j}$$

in which $\alpha^{i_1} = h_{i,j}$ with $i \neq j$.

Notice that in each mode the conditions on which the error type is determined are mutually exclusive.

From these conditions the hardware implementation of the decoder can be derived straightforwardly, as we will see in the next section.

Note that incorrigible errors can be detected in some cases. Otherwise they will be interpreted by the decoder as no error or lead to miscorrection. Because in random mode 157 of the 256 possible syndromes are related to correctable error patterns or no error, the probability that an arbitrary incorrigible error will be detected is only about 38% in random mode.
The decoding principle which has been described in this section for the (4,2) code can be applied for any other (N, K) symbol- and bit-error-correcting code with the same correction capabilities (the detection properties may be better).

2.9 Decoder implementation

In the previous section we dealt with the principles on which the decoding algorithm is based. In this section some details about the decoder design will be presented. Many design decisions that have been taken are influenced by the required minimum decoding time, which should be less than 100 nsec. and the fact that the decoder should be implemented as a CMOS-LSI circuit, (technology 1981). The only solution then is to implement the decoder as a

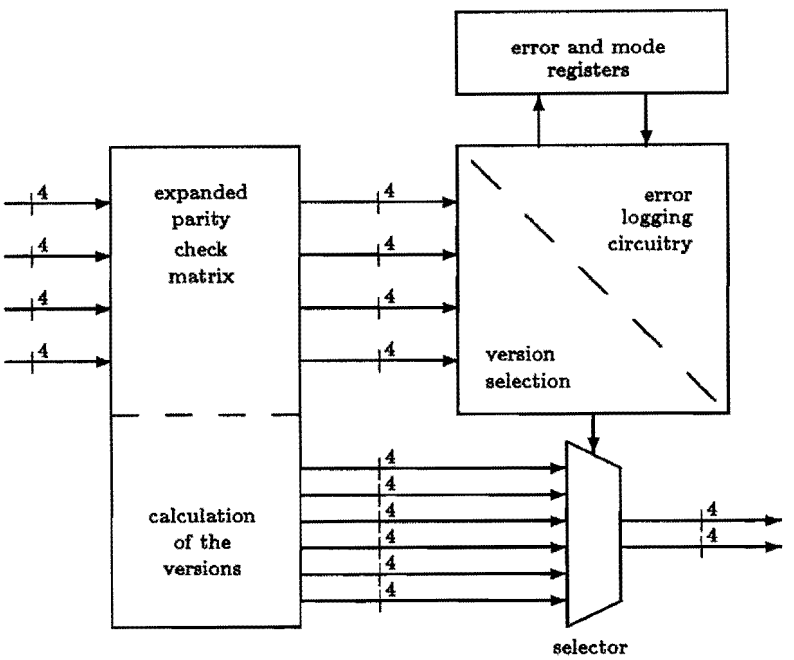


Figure 2.29: *Block diagram of the decoder for the (4,2) single-symbol double-bit-error-correcting code*

combinatorial network. Hence the number of gates was of little importance compared to the depth of the combinatorial network.

The block diagram of the decoder is shown in Figure 2.29. It consists of:

- A straightforward implementation of the expanded parity check matrix.
- Circuitry for the calculation of six versions of the data word. Each version is calculated from a different pair of code word symbols.
- Circuitry for determining, from the syndrome, which of the six versions is (are) correct. This circuitry is combined with the circuitry provided

for error logging and automatic mode register update.

- A selector for selecting the correct version.

This architecture is advantageous because determining which of the symbols of the code word are faulty is done in parallel with the calculation of the versions. Moreover the latter calculations take advantage of the calculation of the syndrome.

Circuitry for the calculation of the syndrome

Racall that the symbols over which the code is defined form a four-dimensinal vector space over $GF(2)$. The basis of this vector space is α^3, α^2, α^1, and α^0. These symbols are associated with the vectors (1000), (0100), (0010), and (0001) respectively. Moreover α is the primitive root of the polynomial x^4+x+1. Hence the correspondence between the symbols denoted by vectors over $GF(2)$ and the symbols denoted as a power of the primitive root of the polynomial x^4+x+1 is as follows:

φ	0 0 0 0	α^7	1 0 1 1
α^0	0 0 0 1	α^8	0 1 0 1
α^1	0 0 1 0	α^9	1 0 1 0
α^2	0 1 0 0	α^{10}	0 1 1 1
α^3	1 0 0 0	α^{11}	1 1 1 0
α^4	0 0 1 1	α^{12}	1 1 1 1
α^5	0 1 1 0	α^{13}	1 1 0 1
α^6	1 1 0 0	α^{14}	1 0 0 1

Multiplication of a symbol $\mathrm{a} = (a_3a_2a_1a_0)$, expressed as a vector over $GF(2)$, with a constant $\alpha^i = (c_3c_2c_1c_0)$ can be represented by a matrix vector multiplication. The multiplication

$$\alpha^i.\mathrm{a}$$

can be represented by

$$\alpha^i.(a_3.\alpha^3 + a_2.\alpha^2 + a_1.\alpha^1 + a_0.\alpha^0)$$

or

$$a_3.\alpha^{i+3} + a_2.\alpha^{i+2} + a_1.\alpha^{i+1} + a_0.\alpha^i$$

Let $\mathbf{c}_j$ be the vector representation of the symbol α^j, then the previous expression can be formulated by:

$$\left(\mathbf{c}_{i+3}^{tr}, \mathbf{c}_{i+2}^{tr}, \mathbf{c}_{i+1}^{tr}, \mathbf{c}_{i}^{tr}\right) \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

The matrix over GF(2)

$$\left(\mathbf{c}_{i+3}^{tr}, \mathbf{c}_{i+2}^{tr}, \mathbf{c}_{i+1}^{tr}, \mathbf{c}_{i}^{tr}\right)$$

is called the companion matrix of α^j.

So for example the multiplication of $(a_3a_2a_1a_0)$ with α^6 is represented by:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

The addition modulo 2 corresponds to the EXCLUSIVE-OR in Boolean algebra, so the hardware implementation of such a multiplication is obvious, see Figure 2.30.

Circuitry for the calculation of six versions of the data word

The six versions of the data word are calculated by determining the inverse of all six 2×2 submatrices of the generator matrix.
So for example the version which follows from c_3 and c_2 is found as follows: From the definition of the generator in (2.15) we know:

$$\begin{pmatrix} d_1 \\ d_0 \end{pmatrix} = \begin{pmatrix} \alpha^0 & \varphi \\ \alpha^7 & \alpha^{11} \end{pmatrix}^{-1} \cdot \begin{pmatrix} c_3 \\ c_2 \end{pmatrix}$$

or

$$\begin{pmatrix} d_1 \\ d_0 \end{pmatrix} = \begin{pmatrix} \alpha^0 & \varphi \\ \alpha^{11} & \alpha^4 \end{pmatrix} \cdot \begin{pmatrix} c_3 \\ c_2 \end{pmatrix}$$

By replacing the symbols of the matrix by their companion matrix and the symbols in the vectors by their vector representation, we obtain a representation of the multiplication which can easily be implemented in binary logic.

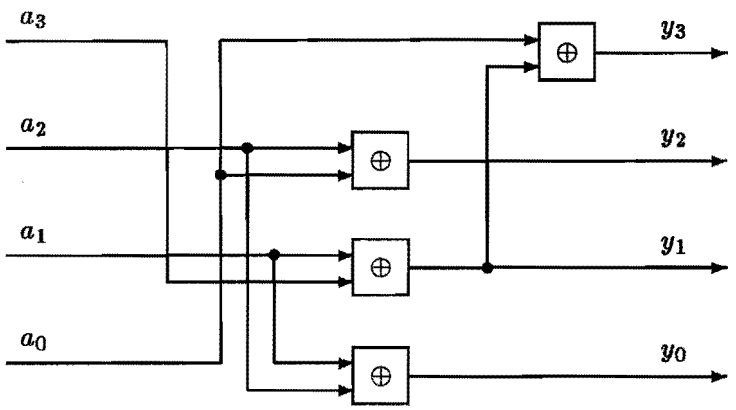


Figure 2.30: *The hardware implementation of the multiplication* $\mathbf{y} = \alpha^6.\mathbf{a}$, *where* α *is a root of the primitive polynomial* $x^4 + x + 1$ *which generates* $GF(2^4)$, *and where* $\mathbf{y} = \langle y_3 y_2 y_1 y_0 \rangle$ *and* $\mathbf{a} = \langle a_3 a_2 a_1 a_0 \rangle$.

For example the calculation of the version which follows from c_3 and c_2 is described as follows:

$$\begin{pmatrix} d_{1,3} \\ d_{1,2} \\ d_{1,1} \\ d_{1,0} \\ \\ d_{0,3} \\ d_{0,2} \\ d_{0,1} \\ d_{0,0} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ & & & & & & & \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_{3,3} \\ c_{3,2} \\ c_{3,1} \\ c_{3,0} \\ \\ c_{1,3} \\ c_{1,2} \\ c_{1,1} \\ c_{1,0} \end{pmatrix}$$

Circuitry for determining, from the syndrome, which of the versions of the data word are correct

From the four 4-bit syndrome symbols the following binary values are derived:

The values $ze(3)$, $ze(2)$, $ze(1)$, and $ze(0)$ are determined by

$$ze(i) \Longleftrightarrow (s_i = \varphi)$$

So in Boolean algebra

$$ze(i) = \overline{s_{i,3}} \cdot \overline{s_{i,2}} \cdot \overline{s_{i,1}} \cdot \overline{s_{i,0}}$$

The 12 logical values $in(i,j)$ with $i \neq j$, $i,j \in \{0,1,2,3\}$, and $h_{i,j} = \alpha^{i_1}$ are defined by:

$$in(i,j) \Longleftrightarrow (s_i \in \{\alpha^{i_1}, \alpha^{i_1+1}, \alpha^{i_1+2}, \alpha^{i_1+3}\})$$

The faulty modules can be selected from these logical values. This will be elucidated for two typical cases, i.e. the selection conditions for the version obtained from c_3 and c_2, and the selection conditions for the version obtained from c_3 and c_1.

If in random mode all symbols of the code word are correct we choose the version of the data word which is obtained from c_3 and c_2. Similarly if in erasure mode 0 or erasure mode 1 no bit-fault occurs, we also choose the version of the data word which is obtained from c_3 and c_2.

With these choices the conditions on which a particular version must be selected can be easily found from the conditions for the error locations as they are stated in the previous section.

The version which is obtained from c_3 and c_2 is selected if:
In random mode

$$(ze(0) \wedge ze(1) \wedge ze(2) \wedge ze(3)) \vee (in(0,1) \wedge in(1,0))$$

In erasure mode 0

$$in(0,1) \vee ze(0)$$

In erasure mode 1

$$in(1,0) \vee ze(1)$$

The version which is obtained from c_3 and c_1 is selected if:
In random mode

$$(in(0,2) \wedge in(2,0))$$

In erasure mode 0

$$in(0,2)$$

In erasure mode 2

$$in(2,0)$$

The other selection circuitry is derived in the same way.

2.10 Some facts about the implementation of a $(4,2)$-concept

When a fault occurs it is masked by the code. These faults have to be reported, although this does not have to take place immediately. In each decoder, therefore, a fault register is available into which the data of the fault are written as soon as the fault occurs. Such fault registers should be regarded as single, unprotected I/O devices which the system must be able to read out. The problems that might be introduced by single unprotected I/O devices are extensively discussed in Chapter 1.

When a symbol and bit-error-correcting code is used, each decoder must be equipped with a mode register which tells the decoder which strategy should be used in decoding an erroneous code word.
The number of different modes depends on the code applied. In the case of a $(4,2)$-concept a random mode, four erasure modes and six single modes can be distinguished, (cf. Section 2.6). In single mode operation the system runs on two modules only. All mode transitions are performed under software control via the decoders, except the transition from random mode to erasure mode which can also be done by the decoders themselves.

The four modules operate in full synchronism. Each of the four modules is provided with its own clock. These four clocks synchronize one another in a fault-tolerant way.
The decoders are combinatorial logic networks: thus at the instant an input changes its logical value the output will be incorrect even if the changing input bits are restricted to a single symbol. Therefore each decoder has to be implemented with an input register which samples the data on the bus.

Synchronous clocks do not guarantee instruction synchronism. Therefore for initial system start-up, resynchronization and reanimation of a repaired

module a special hardware mechanism is required which forces the system into instruction synchronism. Because most microprocessors can only be started in a predefined way by applying a reset, this will form the basis of the instruction synchronization.

The recovery procedure is implemented as follows:

- All information stored in the (correct functioning) microprocessors is saved by transferring it to fixed memory locations.
- In each (correct) module a reset signal is generated, initiated by software and sent to all other modules.
- In each module a majority vote is taken on these signals by which the hardware reset mechanism is activated.
- The microprocessor data are retrieved from the fixed memory locations (via the decoders, so faulty data is corrected).

The whole procedure described above must be implemented as an indivisible action. However, updating the memory of the repaired module can be done at any moment by just letting the system read and write back memory words.

Chapter 3

A class of algorithms for reaching interactive consistency based on voting and coding

In this chapter a new class of synchronous deterministic algorithms for reaching interactive consistency will be presented. The number of modules and the number of rounds of information exchange of these algorithms is minimal, i.e. they meet both the $N = 3T + 1$ bound and the $K = T + 1$ bound. The class of algorithms will be based on error-correcting codes and comprises the original algorithm based on voting published in the early eighties. This new class of algorithms based on voting and coding requires considerably less data communication than the original algorithm. Moreover the class of algorithms also comprises algorithms based entirely on voting which require considerably less data communication than the original algorithm.

The algorithm based on voting and coding will be defined and proved on the basis of a class of algorithms, called the Dispersed Joined Communication algorithms.

3.1 Introduction to the Byzantine Generals Algorithms

3.1.1 The definition of the Byzantine Generals problem

In the introduction to the Input Problem, Section 1.5, it already has been explained that fault-tolerant systems will always be connected to other systems based on different methods for reliability improvement and in any case will be connected to basically unreliable input devices.
These unreliable sources might cause a fault-tolerant system to break down even if the fault-tolerant system does contain no more faults than it is designed to tolerate. The example in Section 1.5 showed that the root of the problem was in the broadcasting of the data by the external source to the N modules of the fault-tolerant system.

In Chapter 2 the definition of (X, Y, T) and (X, Y, Z, T) fault-tolerance has been based on the assumption that the function X can be performed fault free if at most T modules are faulty.
The Input Problem is related to the Byzantine Generals Problem, or as it is sometimes called the Interactive Consistency Problem. This Interactive Consistency Problem will be the subject of this chapter. In Chapter 5 we will show that the Input Problem indeed can be solved by means of an Interactive Consistency Algorithm or similarly by means of Dispersed Joined Communication algorithms.

The Interactive Consistency Problem is considered to be one of the most important problems in distributed computing. The problem is, in one of the first papers on this topic by Lamport, Shostak and Pease[Lamp 82], sketched as follows:

We imagine that several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messengers. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. So the generals must have an algorithm to guarantee that:

A. All generals decide upon the same plan of action.

The loyal generals will all do what the algorithm says they should, but the traitors may do anything they wish. The algorithm must guarantee condition **A** regardless of what the traitors do.
The loyal generals should not only reach agreement, but should agree upon a reasonable plan. We therefore also want to insure that

B. A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Here we finish quoting the sketch of the problem in paper of Lamport et al. [Lamp 82]
Condition **B** is hard to formalize, since it requires a definition of a bad plan. Therefore the problem sketched above will be divided into the following two distinct parts

- Initially, all generals possess a part of the data, the initial data, on which the plan will be based. This data will be distributed by all generals to all generals by an algorithm, which is called *the Interactive Consistency Algorithm.* This algorithm ensures that :
  - All loyal generals agree among each other on the data they think they have received from one of the generals, and
  - if the latter is a loyal general the above-mentioned agreement should equal the initial data actually sent by this general.
- After having applied this algorithm on the initial data possessed by each of the generals, all loyal generals will be in possession of the same data on which they apply the same algorithm, whatever this may be, in order to come to a good plan.

Hence the problem boils down to the design of an *Interactive Consistency Algorithm* or *Byzantine Generals algorithm.*

3.1.2 The parameters relevant for interactive consistency algorithms

The prime parameters that characterize an interactive consistency algorithm are:

- The number of generals, N.

- The maximum number of traitors, T, that can be tolerated for the algorithm fulfilling its requirements. If an algorithm can tolerate at most T traitors, it is said to be *T-resilient.*

- The number of *rounds* K, i.e. the maximum number of times a message is relayed from the one general to the other, or in other words, the depth of the algorithm.

- The availability of a path between a pair of generals, i.e. the graph which represents the communication possibilities.

- The total amount of data which has to be transmitted between the generals.

There are however several other parameters that characterize the Byzantine Generals Problem. These parameters define the synchrony of the system, the behaviour of the traitors, and the way in which the algorithm terminates.

By the synchrony of the system we mean whether a loyal general responds within a commonly known time span or not. If the loyal generals relay the data received, possibly after having processed it, within a commonly known limited time span, traitors refusing to relay data can be detected by the loyal generals by means of a time-out mechanism. Such systems will be called *synchronous systems.*

The second parameter mentioned is the behaviour of the traitors. In general they can act as they wish, such as refusing to relay data, mutilating the data or sending conflicting information to different destinations. However, the behaviour of the traitors is limited if the initial data to be broadcasted by a general is enciphered and signed in such a way that the other generals can decode the message but not encipher and sign it again. In this case a traitor only can refuse to relay the message or mutilate it. The latter however will immediately be detected by the receiver. This divides the algorithms in algorithms with and without *authentication.*

In non-authenticated algorithms, a general at least needs to know by which general the messenger was sent, i.e. he needs to know the sender of the message he receives, regardless of whether the message is correct or not.

One of the prime parameters is the maximum number of rounds K needed by the algorithm. For some algorithms this number is fixed or always less than some fixed constant; these algorithms are called *deterministic*. However, other algorithms might need an infinite number of rounds. In that case K is a random variable with an finite estimated value. The latter algorithms are called, for more than this reason alone, *randomized Byzantine Generals protocols*.

The last parameter to be mentioned is the *connectivity* of the graph representing the communication possibilities between the generals. We already pointed out in the sketch of the Byzantine Generals problem that the generals communicate with one another by means of messengers. Some of these messengers might not be available at all and the communication between two generals can only be done via a third general. The communication possibilities between the generals can be expressed by a graph in which the nodes are the generals and an edge denotes an existing direct communication possibility between two generals. It is readily seen that this graph needs to have a minimum connectivity in order to fulfil the interactive consistency requirements.

In the following we will abstract from the story of the generals and consider the Byzantine generals problem as the problem of N communicating modules with independent data links between the modules. Among these modules T or less are behaving maliciously, possibly by transmitting conflicting information to different parts of the network, i.e. generating broadcast errors. Whenever a module transmits by means of an algorithm a message to all other modules (or possibly conflicting messages when it is malfunctioning), we define that the algorithm fulfils the *interactive consistency* requirements when the following conditions are fulfilled:

Definition 3.1 (Interactive Consistency)
If an algorithm runs on a system consisting of N modules of which one is the source, and if in this system at most T modules behave maliciously, this algorithm is said to fulfil the interactive consistency requirements when the following conditions are fulfilled, regardless of which modules are faulty and what data was sent by the source:

- *The well-functioning modules agree among each other on the data they think they have received from the source.*

- *If the source is well-functioning, the above-mentioned agreement should equal the data actually sent by the source.*

□

3.1.3 Results published

In this section a chronological overview will be presented of the most important papers which address the topic of interactive consistency. Many papers are not mentioned because they only deal with a related or weaker problem, or these papers are overruled by improved ones. Some of these papers are however mentioned in the list of references.

The first publication in which a problem was addressed which is closely related to the Byzantine generals, is the paper by Davies and Wakerly, [Davies 78], in 1978, in which the mutual synchronization of modules, some of which produce broadcast errors, was investigated.
The investigation at Stanford Research International of a fault-tolerant computer, called SIFT, based on software implemented masking (cf. reference [Wensley 78]) induced the definition and partial solution of the Byzantine generals problem. The result was a publication by Pease, Shostak and Lamport in 1980, [Pease 80]. In this paper the problem of interactive consistency (IAC) was formulated and it was shown that the IAC requirements as stated in Definition 3.1 for synchronous systems without authentication cannot be fulfilled if the number of faulty modules is one third of the total number of modules or more. For synchronous authenticated systems they showed that IAC only can be obtained if $N \geq T + 1$.

Notice that the interactive consistency requirement can always be fulfilled if $N = 2$ or $N = 1$. Moreover a system in which $N - 1$ modules behave maliciously also satisfies the interactive consistency requirements. So we only consider systems with $N \geq 3$ in which less than $N - 1$ modules behave maliciously.
It is readily seen that the $N \geq 3T + 1$ bound for synchronous algorithms without authentication and the $N \geq T + 1$ for synchronous algorithms with authentication at least must hold for asynchronous systems.

Moreover in [Pease 80], a synchronous algorithm with and a synchronous algoritbm without authentication was presented, in which the above-mentioned

bounds were met. In both algorithms the number of rounds, K, needed by the algorithms is $T+1$. The number of messages to be transmitted by the algorithms grows exponentiall with T, thus the number of messages which must be transmitted is $O(N^T)$.

In papers by Dolev, [Dolev 81], [Dolev 82-1], the connectivity of the graph, which represents the communication possibilities between the modules has been studied. In this paper it is shown that the connectivity of this graph must be at least $2T+1$ for non-authenticated algorithms. A synchronous algorithm which meets this bound is presented, but the number of rounds needed by this algorithm will in general be larger than $T+1$. The latter depends on the graph. The amount of messages to be transmitted between the modules during the execution of the algorithm again is exponential in T.
Notice that the connectivity of the graph in the case of algorithms with authentication trivialy must be at least $T+1$.

In a paper by Lamport, Shostak and Pease [Lamp 82], the IAC problem is studied for a special class of graphs and algorithms are described which solve the problem in these cases.

Synchronous algorithms, in which the number of messages which needs to be transmitted is polynomial in N and T, were first published by Dolev and Strong [Dolev 82-2] and thereafter improved in a paper by Dolev, Fischer, Fowler, Lynch and Strong [Dolev 82-3]. In this paper a synchronous algorithm based on messages without authentication is described, which can be used for any $N \geq 3T+1$, which requires $K = 2T+3$ rounds, and in which the total number of messages which needs to be transmitted is only $O(NT + T^3 \log T)$.

One of the most surprising results which has been obtained is that the minimum number of rounds needed for any synchronous algorithm is $T+1$. This bound holds for both algorithms with and without authentication. The $K \geq T+1$ bound was first published by Fischer and Lynch [Fischer 82] and thereafter it was generalized by Dolev and Strong [Dolev 83-1].

The latter paper [Dolev 83-1] also presented a synchronous algorithm based on authentication, with $N = T+1$ and $K = T+1$, and which only requires $O(NT)$ messages.

So far, all algorithms published are deterministic and are based on a synchronous system. Thus the maximum number of rounds needed by the algorithm is bounded by some constant and any correctly functioning module will pass through its data within a commonly known time span.
Fisher, Lynch and Paterson [Fischer 83], [Fischer 85] showed that in asynchronous systems deterministic algorithms are impossible. These results have been elaborated and strengthened by Dolev, Dwork and Stochmeyer, [Dolev 83-2] by defining more carefully the meaning of asynchrony.

Randomized Byzantine generals algorithms were first published by Rabin, [Rabin 83] and Ben-Or, [Ben-Or 83]. Rabin published an algorithm with $N > 10T$ for asynchronous systems. The expected number of rounds needed by this algorithm is only four. The algorithm published by Ben-Or needs $N > 5T$ modules. The latter result has been improved by Bracha [Bracha 87-1]. This paper proves that asynchronous randomized algorithms are possible if and only if $N > 3T$. For this proof the definition of termination needed to be refined.

In [Bracha 87-2] Bracha presents a randomized algorithm for synchronous systems which only needs $O(\log N)$ expected number of rounds of information exchange. If authenticated messages are used the number of modules must be $N > 2T$. Without authentication $N > 3T$ is required. This is an improvement compared to the $K \geq T + 1$ for deterministic algorithms.

Summary of the results published

The main results obtained in the papers mentioned can be summarized as follows:

- Interactive consistency algorithms
  - without authentication are only possible if $N \geq 3T + 1$, and
  - with authentication are only possible if $N \geq T + 1$.
- The connectivity K, of the graph representing the communication possibilities in the system must be
  - at least $2T + 1$ for non-authenticated messages, and
  - at least $T + 1$ for authenticated messages.

- Deterministic IAC algorithms for asynchronous systems are not possible.

- Deterministic IAC algorithms for synchronous systems are only possible if $K \geq T + 1$.

- Deterministic IAC algorithms based on synchronous systems, a fully connected graph, and the following parameters exist:

  - non-authent., $N \geq 3T + 1$, $K = T + 1$, $\#\text{mess} = O(N^T)$.
  - authent., $N \geq T + 1$, $K = T + 1$, $\#\text{mess} = O(N.T)$.
  - non-authent., $N \geq 3T + 1$, $K = 2T + 3$, $\#\text{mess} = O(N.T + T^3 \log T)$.

- Deterministic IAC algorithms for synchronous systems based on a not fully connected graph can always be derived from the algorithms which are based on a fully connected graph at the expence of the number of rounds that are needed, provided the connectivity requirement is fulfilled.

- Randomized IAC algorithms exist for the following parameters:

  - asynch., non-authent., $N \geq 10T + 1$, $K_{expected} = 4$.
  - asynch., non-authent., $N \geq 3T + 1$
  - synch., non-authent., $N \geq 3T + 1$, $\#\text{mess} = O(\log N)$
  - synch., authent., $N \geq 2T + 1$, $\#\text{mess} = O(\log N)$

Open question:

This summary shows clearly that there are many open questions. In practice the application of Interactive Consistency Algorithms is limited by the large number of messages which needs to be transmitted between the modules of the system, cf. Chapter 4. Therefore one of the most challenging questions is whether a bound can be proven on the minimum uumber of messages which need to be transmitted for obtaining Byzantine agreement given N, K and T.

3.2 Introduction to the algorithms and their proof

3.2.1 A survey of the algorithms considered

In the following we will present a class of synchronous deterministic algorithms which solves the Byzantine Generals problem for all values $N \geq 3T+1$ and $K \geq T+1$. This class of Interactive Consistency Algorithms which is based on voting and error-correcting codes will be derived from a class of algorithms which we will call Dispersed Joined Communication algorithms. The latter class of algorithms satisfies more liberal properties than those which are required for the Interactive Consistency Algorithms.

The basic ideas behind these classes of algorithms have similarities with the algorithm which we will call the Pease algorithm and which is presented in [Pease 80], Dolev [Dolev 82-1], and van Gils [Gils 85] and with the (X, Y, Z, T) fault-tolerant systems.
The Pease algorithm is a member of our class of algorithms.

In the new class of IAC algorithms presented in this chapter, the amount of messages which needs to be transmitted between the modules is reduced considerably compared with the existing synchronous deterministic non-authenticated algorithms.

- Firstly by reducing the number of directions in which a message is forwarded and
- secondly by replacing the broadcast functions by the encoder functions of error-correcting codes and simultaneously replacing the voting function in the decision-making process by the decoder functions of the error-correcting codes applied in the broadcast process.

The class of Interactive Consistency Algorithms which is the issue of this chapter thus contains a subclass which is based on voting, cf. Figure 3.1. In this subclass the only functions applied are broadcasting data and majority votes on data. In the entire class of algorithms, however, the broadcast function is replaced by the encoding function of an error-correcting code and the majority vote by the corresponding decoder function of the error-correcting code.
The difference between the algorithms in the subclass based on voting and the algorithm presented in [Pease 80] is that in the latter algorithm during

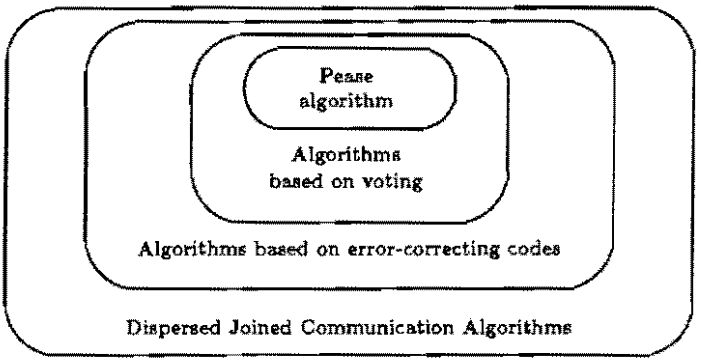


Figure 3.1: *The classes of Interactive Consistency Algorithms and the class of Dispersed Joined Communication Algorithms discussed in this chapter.*

each round a message is relayed to all modules which have not yet passed by the message, while in our algorithm each message only needs to be sent to at least $2T + 1$ modules which have not yet been passed.

3.2.2 The way in which the algorithms are described

Let a system be composed of N fully interconnected modules. So between any two modules a communication channel is available in both directions. Due to these individual channels between the modules, if it behaves correctly a receiving module always 'knows' which module has inserted data on the input side of the channel, regardless of whether the sending module behaves correctly or maliciously.
The modules are identified by the elements of the set **Ns**. So $|\mathbf{Ns}| = N$.
At most T of the modules in the system are allowed to behave maliciously.

In the current context an algorithm aims at transmitting a message from a particular source to a number of destinations. The algorithm prescribes the rules and formats for conducting communications on the network and the operations performed on the messages in the modules. So the description of an algorithm may be regarded as a protocol.
On the other hand, the input of an algorithm is the original message in the source and the output are the decisions (estimates) calculated by the destinations about what the source tried to send to them.

So an algorithm has both topological and behavioural aspects, i.e.:

- Topological aspects which provide information about the source module, the set of destinations and the way a message is routed through the network from the source to the destinations.
- Behavioural aspects which provide information about the functions performed in the modules on the messages before they are forwarded to the next module and the functions which are performed in order to calculate the decision in the destinations.

We restrict ourselves to synchronous deterministic algorithms. This means that the modules in the system run synchronously and have a common notion of time. Moreover the number of rounds of the algorithm is fixed. Therefore the system can be modelled according to the Moore model. The number of time instances used by the algorithm is $K+1$, hence the algorithm comprises $K+1$ rounds synchronously executed by the modules. These rounds are enumerated $0, 1, \cdots, K$.
For convenience an algorithm is divided into two parts:

- a *broadcasting process* which comprises the rounds $0, \cdots, K-1$.
- a *decision-making process* which is executed during round K and in which the result of the algorithm in each module is calculated.

Each round of the broadcasting process consists of two parts:

1. A number of (combinatorial) functions is applied on the data received in the previous round in each module, one for each module to which a result will be sent. Such a function may depend on:
   - the module in which it is executed,
   - the round, and
   - the destination to which the result is sent.
2. The exchange of data between the modules.

During the last round, i.e. the decision-making process, no information exchange takes place. The algorithm thus comprises K rounds of information exchange.

The broadcasting process

Data sent from one module to another during a particular round is divided into a number of *messages*. Each message is treated by a module individually. Functions are thus only applied on a single message and the result of the function application again is a single message. Clearly in a particular module during a particular round more than one function may be applied on the same incoming message resulting in more than one outgoing message, but during that round all these messages will be sent in different directions. This means that a message which is received during a particular round can only cause single messages to be sent in different directions. Conversely, if a message is transmitted from a first module to a second, then this is always caused by a message which in the previous round has been received by the first module. Thus no messages are generated spontaneously. Also the original message available in the source at the beginning of round 0 is regarded as having been received during the previous round.
Consequently the messages can be identified by the path they travelled through the network. For example, a message which originated in module a and which after modification by a function was sent to module b, and again after modification was sent to module c, is identified by the string (path) (a, b, c). Only $K + 1$ time instances are taken into account which encompass K rounds of information exchange. So messages are identified by elements of the set of all strings of length $K + 1$ or less over the set of module identifiers $\mathbf{Ns}$. A string will be denoted by a symbol like $\underline{s}$.
Messages are not forwarded in all directions, therefore we define a function $\mathbf{B}$ on the set of all strings of length K or less over $\mathbf{Ns}$, such that $B(\underline{s})$ is the set of modules to which the message which is identified by $\underline{s}$, is sent after modification.

The messages, or more precisely the values of the messages, are represented as a function m on the set of message identifiers. So $m(\underline{s})$ is the message which is identified by $\underline{s}$ and $m(a, b, c)$ is the message (= message value) which is received during round 1 by module c from module b and which travelled from module a via module b to module c.

The decision-making process

The decision-making process acts independently in each module on the messages which are received by that module during the entire broadcast process.

If we are dealing with Interactive Consistency Algorithms, the results of the calculations performed in each module by the decision-making processes have to fulfil the interactive consistency requirements as defined in Definition 3.1 on Page 119.

3.3 The Dispersed Joined Communication Algorithms

3.3.1 Introduction

In this section we will define a new class of algorithms which will be called Dispersed Joined Communication algorithms (DJC algorithms). These aim to transmit a message from a single source module to a number of destinations in the presence of a number of maliciously behaving modules.

In order to be able to tolerate modules which behave maliciously, the communication between the source and the destinations is *dispersed*, i.e. the message which needs to be transmitted is sent possibly in differently modified versions, via different paths from the source to the destinations.
A DJC algorithm prescribes the way in which the message is forwarded through the network from the source to the destinations, the way in which the messages are modified by the modules, and the way in which the final result is calculated in the destinations. Although the message passing is dispersed, the message passing and modification for different destinations is *joined* as much as possible, i.e. for any two destinations d and e, it holds that all paths from the source to these destinations are shared as much as is compatible with the additional requirement that a message is never relayed to a module that it has already passed. This means that the set of directions $\mathbf{B}(\underline{s})$ into which a module will forward a message $\underline{s}$ it received in the previous round does not depend on its final destinations, but only on the path followed hitherto.

First in Section 3.3.2, we will describe the way in which these algorithms are constructed.
Next, in Section 3.3.3, we will investigate for which parameters N, K and T they can be constructed, i.e. we will investigate the topological aspects of the algorithm.
And finally in Section 3.3.4, we will prove some of their behavioural prop-

erties, i.e. the relation between the original message value in the source module and the value finally calculated in the destinations.

The behavioural properties of the DJC algorithms have strong similarities with the interactive consistency requirements but differ from the latter that in particularly well-defined circumstances correctly functioning modules might arrive at different decisions.

To be more precise, the DJC algorithms will be defined such that they satisfy the following behavioural properties:

- If the source and destination are both functioning correctly, then the decision calculated by the decision-making process in the destination equals the original message in the source.

- For an algorithm which is based on K rounds of information exchange and which aims at communicating a message from a source module a to a number of destinations, it will hold that if the result calculated in two correctly functioning destinations is different then a message has travelled along a path of length K from the source module a to these destinations consisting of K different modules which all (the source module a inclusive) behave maliciously.

A sub-class of these DJC algorithms will define and prove the properties of a new class of Interactive Consistency Algorithms which are based on voting and coding.
Moreover the algorithms which solve the Input Problem will be based on these DJC algorithms.

So the class of DJC algorithms encompasses the class of Interactive Consistency algorithms which are based on voting and coding, cf. Figure 3.1 on page 125.

Dispersed Joined Communication Algorithms are defined on a set Ns of fully interconnected modules. A particular DJC algorithm aims at sending a particular message from a particular *source module* a to a particular *set of destinations* D, by means of K rounds of information exchange, Figure 3.2.

In general there will exist many DJC algorithms which have the same properties.

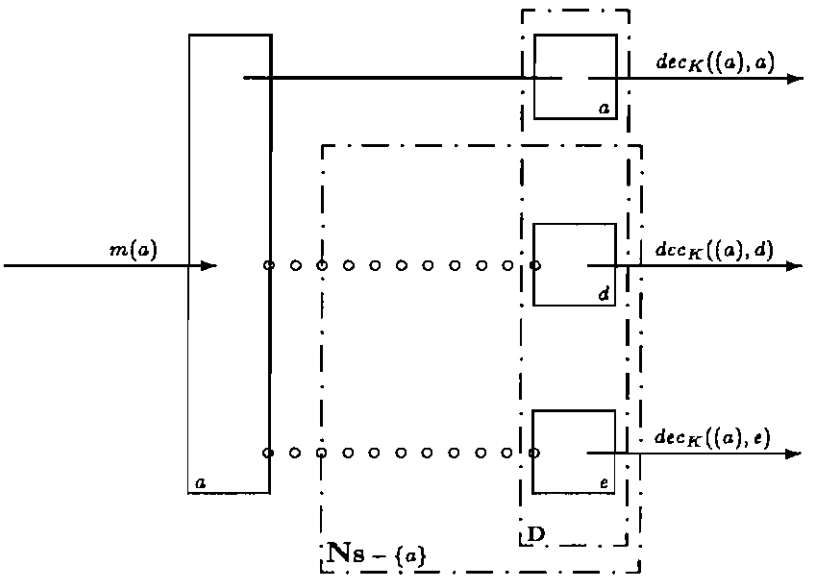


Figure 3.2: *A pictural representation of an algorithm in the class* $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$.
Direct communication is indicated by ——, *communication via other modules by* o o o.

Therefore we define classes

$$\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns}) \tag{3.1}$$

of DJC algorithms in which:

T is the maximum number of maliciously behaving modules which is tolerated.
K is the number of rounds of information exchange.
a is the source module of the algorithm.
$\mathbf{D}$ is the set of destinations.
$\mathbf{Ns}$ is the set of modules in the system.

Obviously these classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ of DJC algorithms are only defined if

$$K \geq 1 \quad \text{and} \quad a \in \mathbf{Ns} \quad \text{and} \quad \mathbf{D} \subset \mathbf{Ns} \tag{3.2}$$

In order to exclude some pathological classes we additionally require

$$|\mathbf{D}| \geq K + 1 \tag{3.3}$$

A particular algorithm in a class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ lays down in detail the way in which a message travels from the source a to the destinations d in $\mathbf{D}$ via different parallel paths in K rounds of information exchange. Moreover the algorithm prescribes the way in which the messages are modified during their journey though the network and the way in which in each destination d a decision is calculated starting from all data received by d.

An algorithm in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ forwards the original message in the source module in K rounds of information exchange to the destinations in $\mathbf{D}$. In accordance to our remarks in the previous section, the original message in the source is denoted by

$$m(\underline{\mathbf{s}}, a) \quad \text{or by} \quad m(a) \tag{3.4}$$

The prefix $\underline{\mathbf{s}}$ to the source module identifier a is only used if we need to distinguish between different messages in the same module a and in that case denotes the path along which the message travelled to module a.

If a message $m(\underline{\mathbf{s}}, a)$ (or $m(a)$) is sent to the modules in the set $\mathbf{D}$ by means of an algorithm from the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$, then the results calculated in the modules d are denoted by

$$dec_K((\underline{\mathbf{s}}, a), d) \quad (\text{or by} \quad dec_K((a), d)\ ) \quad \text{with} \quad d \in \mathbf{D} \tag{3.5}$$

Notice that for the definition of the algorithms it is not necessary to have an expression for a particular algorithm in a class of algorithms. Moreover we do not use an explicit single expression for the relation between the input value $m(a)$ and the output value $dek_K((a), d)$.
In order to elucidate the difference between an explicit and an implicit description, we consider the following example.
Suppose a class of systems S consists of two concatenated modules identified by a and b. The input value to a is denoted by x, the value sent from a to b

by y, and the output value of b is denoted by z. In module a a function f is executed which belongs to a class of functions F and in module b a function g is executed which belongs to a class of functions G.
The explicit definition of the class of systems S is as follows:

> Let S be a class of systems with input value x and output value z, the relation between x and z is defined by $z = s(x)$ in which $s = g \circ f$ and f is a function from the class F and g is a function from the class G.

The implicit definition of the class of systems S is as follows:

> Let S be a class of systems with input value x and output value z. The systems in the class S are composed of two modules a and b. In module a a function from the class F is executed on its input value x and the result is forwarded to module b. In module b a function from the class G is executed on the value received from a. The result is called z.

Both the explicit and the implicit description define the behavioural aspects of the system equally well. However, the latter (implicit) definition provides additional topological information about the system. This topological information is important for the DJC algorithms and therefore we will apply the implicit way of defining the DJC algorithms.

3.3.2 The construction of the Dispersed Joined Communication Algorithms

Let a system be composed from a number of fully interconnected modules. At most T of the modules in the system are allowed to behave maliciously.

The algorithms in the classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ will be defined recursively with respect to K. The basis of the recursion is the case $K = 1$.

The construction of the algorithms in the class $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$

An algorithm in the class $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ is based on only one round of information exchange.
Recall from (3.1) through (3.3) that a denotes the source module of the algorithm, $\mathbf{D}$ denotes the set of destinations and $\mathbf{Ns}$ denotes the set of modules in the system.

Moreover recall that classes $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ of DJC algorithms are only defined if

$$a \in \mathbf{Ns} \quad \text{and} \quad \mathbf{D} \subset \mathbf{Ns} \quad \text{and} \quad |\mathbf{D}| \geq 2 \tag{3.6}$$

These restrictions concern the topological aspects of the class of algorithms.

Under the given conditions the class $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ contains the following algorithm:
During round 0, the source module a sends the original message $m(a)$ directly and unchanged to all modules in $\mathbf{D} - \{a\}$. If $a \in \mathbf{D}$ then module a keeps a copy of the message $m(a)$ itself in order to be used in the decision-making process during round 1. The messages received by the modules d in $\mathbf{D} - \{a\}$ from module a are denoted by $m(a, d)$, cf. Figure 3.3.

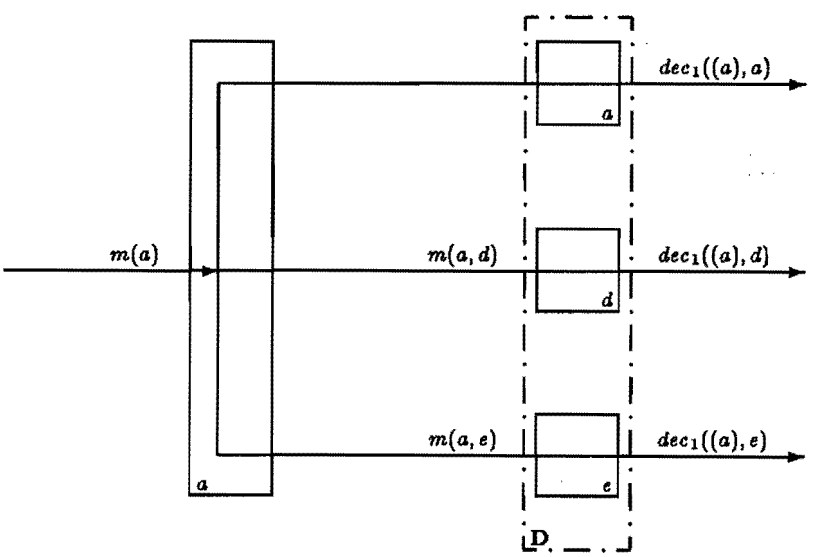


Figure 3.3: *A pictural representation of an algorithm in the class* $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$.

During round 1 the decision-making process is executed in which in each module d, with $d \in (\mathbf{D} - \{a\})$, the message $m(a,d)$ received from a is taken as decision $dec_1((a),d)$ and if $a \in \mathbf{D}$ then the decision $dec_1((a),a)$ in module a will be equal to the stored message $m(a)$.

So the behavioural aspects of the algorithms in the class $\mathcal{A}(T,1,a,\mathbf{D},\mathbf{Ns})$ (starting from correctly functioning modules) are defined by:

$$\begin{gathered} d \in (\mathbf{D} - \{a\}) \Longrightarrow m(a,d) = m(a) \\ a \in \mathbf{D} \Longrightarrow dec_1((a),a) = m(a) \\ d \in (\mathbf{D} - \{a\}) \Longrightarrow dec_1((a),d) = m(a,d) \end{gathered} \tag{3.7}$$

Notice that for any module a and any sets $\mathbf{D}$ and $\mathbf{Ns}$ which satisfy (3.6), such an algorithm can be constructed. Thus the class $\mathcal{A}(T,1,a,\mathbf{D},\mathbf{Ns})$ in that case is non-empty.

The recursive construction of the algorithms in the class $\mathcal{A}(T,K,a,\mathbf{D},\mathbf{Ns})$ with $K > 1$ in terms of algorithms from the set of classes $\mathcal{A}(T,K-1,b,\mathbf{D}-\{a\},\mathbf{Ns}-\{a\})$ with $b \in \mathbf{Ns}$.

Bear in mind that the original message in a source module is denoted by $m(a)$. Also remember that the result of a DJC algorithm from the class $\mathcal{A}(T,K,a,\mathbf{D},\mathbf{Ns})$ calculated in a module d with $d \in \mathbf{D}$ is represented by $dec_K((a),d)$.

The construction of the algorithms in the class $\mathcal{A}(T,K,a,\mathbf{D},\mathbf{Ns})$ with $K > 1$ will be based on encoding the original message $m(a)$ in the source into symbols of a T-error-correcting code, thereafter transmitting each symbol to a different module b, which forwards the received symbols to the destinations by means of an algorithm from the class $\mathcal{A}(T,K-1,b,\mathbf{D}-\{a\},\mathbf{Ns}-\{a\})$.

Therefore let $\mathcal{Y}_{(a)}$ be the encoder function of some T-error-correcting code of which the code words consist of $n_{(a)}$ symbols of size $b_{(a)}$ and of which the data words consist of $k_{(a)}$ symbols. The corresponding decoder function is as usual denoted by $\mathcal{Y}_{(a)}^{(-1)}$. Suppose the original message $m(a)$ consists of $k_{(a)}$ symbols and the symbol which is sent to b is denoted by $m(a,b)$, then

this symbol is related to $m(a)$ by

$$m(a,b) = y_{(a)}(b)(m(a)) \tag{3.8}$$

in which the $y_{(a)}(b)$ is called the partial encoder function of the encoder function $y_{(a)}$ which delivers the symbol which has to be sent to module b.

With the preceding remarks we are able to define the DJC algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ with $K > 1$. See Figures 3.4, 3.5 and 3.6.

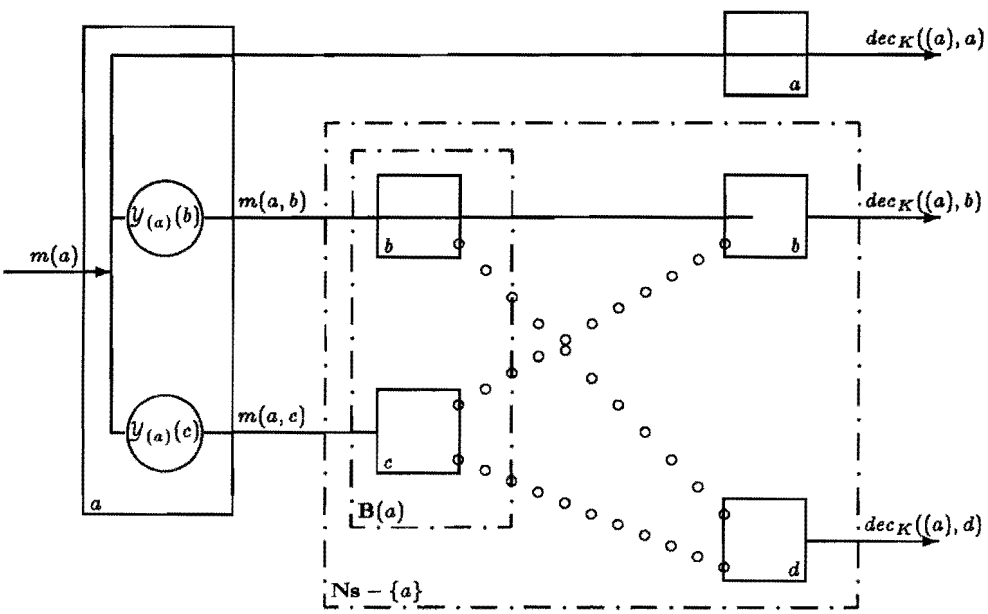


Figure 3.4: *A pictural representation of the construction of an algorithm in the class* $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$.

The DJC algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ with $K > 1$ are constructed as follows:

1. During round 0

   (a) If the source module is one of the destinations, thus $a \in \mathbf{D}$, the original message $m(a)$ in the source is kept stored in the module in

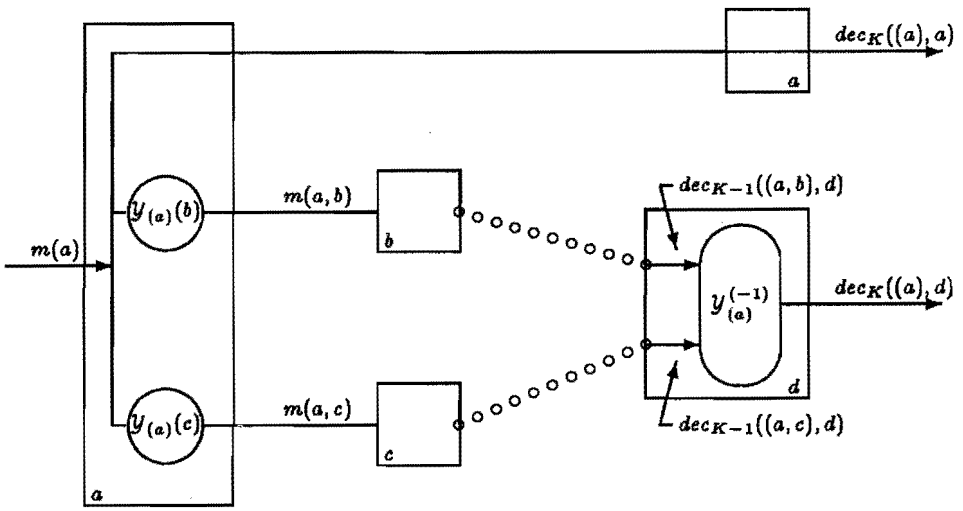


Figure 3.5: *A pictural representation of the way in which the decision* $dec_K((a),d)$ *in a module* d *with* $d \notin \mathbf{B}(a)$ *is obtained from the partially encoded messages sent by module* a *to the modules in* $\mathbf{B}(a)$

order to be used later on during round K in the decision-making process, cf. Figure 3.4.

(b) Furthermore in the source module a a number of partially encoded versions $m(a,b)$ of the original message $m(a)$ are calculated such that $m(a,b) = y_{(a)}(b)(m(a))$.

(c) Thereafter each of these partially encoded versions $m(a,b)$ of the original message is sent to a different module. These modules are indicated by the *next-set* $\mathbf{B}(a)$. So $b \in \mathbf{B}(a)$. The number of modules b to which the partially encoded messages are sent must be at least $2T+1$.

2. During the rounds $1, \cdots K$, each module b in the next-set $\mathbf{B}(a)$ forwards the received partially encoded message $m(a,b)$ to the destinations indicated by $\mathbf{D} - \{a\}$ by means of an algorithm from the class $\mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$. The results of these algorithms in the destinations $d \in (\mathbf{D} - \{a\})$ are denoted by $dec_{K-1}((a,b),d)$. These re-

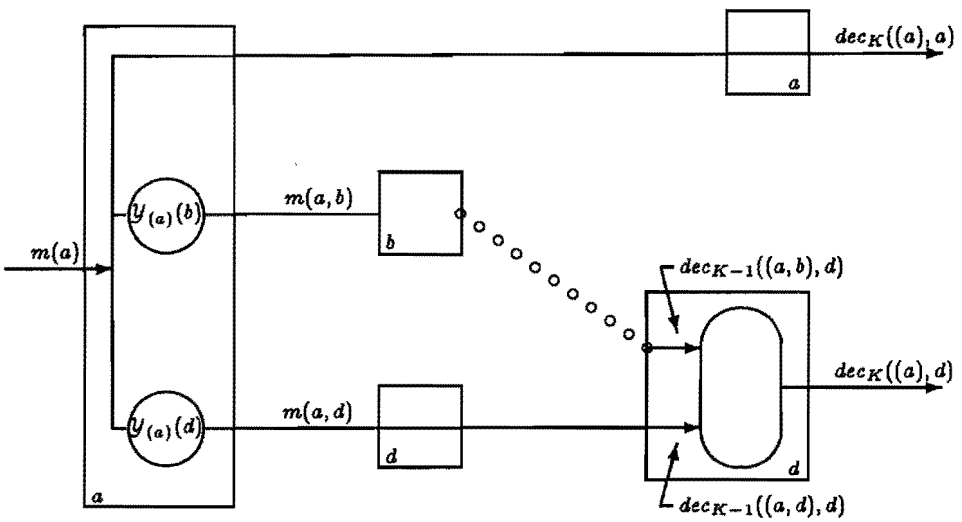


Figure 3.6: *A pictural representation of the way in which the decision* $dec_K((a),d)$ *in a module* d *with* $d \in \mathbf{B}(a)$ *is obtained from the partially encoded messages sent by* a *to the modules in* $\mathbf{B}(a)$

sults are calculated during the first part of the decision-making process which is executed in round K.

3. During the second part of the decision-making process which is executed during round K, the decision $dec_K((a),d)$ in the modules d with $d \in (\mathbf{D} - \{a\})$ is obtained by applying the decoder function $y_{(a)}^{(-1)}$ on the results $dec_{K-1}((a,b),d)$ with $b \in \mathbf{B}(a)$.
If $a \in \mathbf{D}$ then the decision $dec_K((a),a)$ is obtained by taking the message value $m(a)$ which had been kept stored in module a, cf. Figures 3.5 and 3.6.

Notice that the algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ require K rounds of information exchange, while the algorithms in the class $\mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$ require $K-1$ rounds of information exchange. The latter are preceded by the round in which the partially encoded messages are sent from a to b. So round t, with $1 \leq t \leq K$, of the algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ corresponds to round $t-1$ of the algorithms in the class $\mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$. And thus the calculation of the decisions $dec_{K-1}((a,b),d)$ in a module $d \in (\mathbf{D}-\{a\})$ precedes the calculation of the decisions $dec_K((a),d)$ during the same round K.

We further remark that during round t, with $t \geq 2$, a module b in general receives more than one message $m(\underline{\mathbf{s}}_i, b)$, which arrives from the source module a via different paths $\underline{\mathbf{s}}_i$. Each of these messages is dealt with by b separately. If $t \leq K-2$, they are each encoded by means of an encoder function that depends on $\underline{\mathbf{s}}_i$ and thereafter forwarded to destinations which are determined by the set $\mathbf{B}(\underline{\mathbf{s}}_i, b)$ which depends on $\underline{\mathbf{s}}_i$. How module b discriminates between the messages it receives is a matter of implementation and will not be further commented on.

Clearly the construction described above is possible if and only if

- A next-set $\mathbf{B}(a)$ can be found which is a subset of $\mathbf{Ns}-\{a\}$ and which contains at least $2T+1$ modules.

- A T-error-correcting code exists of which the code words consist of $|\mathbf{B}(a)|$ symbols.

- The classes $\mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$ of DJC algorithms with $b \in \mathbf{B}(a)$ are all non-empty.

(3.9)

The behavioural aspects of the algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ can

be summarized as follows:

$$
\begin{aligned}
& m(a,b) = y_{(a)}(b)(m(a)) \quad \text{for all} \quad b \in \mathbf{B}(a) \\
& dec_{K-1}((a,b),d) \quad \text{follows from} \quad m(a,b) \quad \text{based on} \\
& \qquad \text{an algorithm from the class} \quad \mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\}) \\
& d \neq a \Longrightarrow dec_K((a),d) = y_{(a)}^{(-1)} \quad \text{applied on} \\
& \qquad\qquad \text{the values} \quad dec_{K-1}((a,b),d) \quad \text{with} \quad b \in \mathbf{B}(a) \\
& d = a \Longrightarrow dec_K((a),d) = m(a)
\end{aligned}
\tag{3.10}
$$

3.3.3 The existence of Dispersed Joined Communication Algorithms in the classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$

The next step is to investigate for which parameters DJC algorithms can be constructed or stated in a different way, for which parameters the classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ are non-empty.

Recall from (3.2) and (3.3) that the classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ of DJC algorithms are only defined if

$$
K \geq 1 \quad \text{and} \quad a \in \mathbf{Ns} \quad \text{and} \quad \mathbf{D} \subset \mathbf{Ns} \quad \text{and} \quad |\mathbf{D}| \geq K+1 \tag{3.11}
$$

If these constraints are not satisfied a class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ is empty by definition.

Within this context, the next theorem will show that the non-emptiness of the classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ only depends on the number of modules in the system, i.e. $|\mathbf{Ns}|$, the number of rounds K and the number T of faults which is to be tolerated.

Theorem 3.1

- *For all T with $T \geq 1$ and*
- *for all K with $K \geq 2$ and*

- *for all fully interconnected systems consisting of* $|\mathbf{Ns}|$ *modules, and*
- *for all source modules* a *in the system, and*
- *for all sets* $\mathbf{D}$ *of destinations, with* $\mathbf{D} \subset \mathbf{Ns}$ *and* $|\mathbf{D}| \geq K+1$

it holds that the class of algorithms $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ *is non-empty if and only if*

$$|\mathbf{Ns}| \geq 2T + K$$

□

Proof:

The theorem will be proved by induction with respect to K. The basis of the induction will be $K = 2$.
However before we elaborate on the classes of algorithms with $K = 2$ and prove the theorem for $K = 2$ we will first have to determine the constraints for the classes of algorithms with $K = 1$.

Throughout the proof we assume that the parameters K and T, the source module a, and the sets $\mathbf{Ns}$ and $\mathbf{D}$ satisfy

$$T \geq 1 \quad \text{and} \quad K \geq 1 \quad \text{and} \quad a \in \mathbf{Ns} \quad \text{and} \quad \mathbf{D} \subset \mathbf{Ns} \quad \text{and} \quad |\mathbf{D}| \geq K+1 \tag{3.12}$$

Recall that classes $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ which do not satisfy these constraints are assumed to be empty.

From the construction of the algorithms in the classes $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ we know that in these algorithms the original message in a is sent directly and unchanged to the modules in $\mathbf{D} - \{a\}$ and that if $a \in \mathbf{D}$ the message $m(a)$ is kept stored in the module. Because we start from a set of fully interconnected modules, such algorithms can always be constructed and thus the class of algorithms $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ is non-empty if and only if the general constraints expressed in (3.11) are fulfilled, i.e.:

$$\mathbf{D} \subset \mathbf{Ns} \quad \text{and} \quad |\mathbf{D}| \geq 2 \tag{3.13}$$

From this immediately follows the necessary and sufficient requirement

$$|\mathbf{Ns}| \geq 2 \tag{3.14}$$

The proof of the theorem for $K = 2$ is as follows:
From our remarks (3.9) on the construction we recall that the construction of algorithms with $K = 2$ is possible if and only if

1. A next-set $\mathbf{B}(a)$ can be found which is a subset of $\mathbf{Ns} - \{a\}$ and which contains at least $2T + 1$ modules.

2. A T-error-correcting code exists of which the code words consist of $|\mathbf{B}(a)|$ symbols.

3. The classes $\mathcal{A}(T, 1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$ of DJC algorithms with $b \in \mathbf{B}(a)$ are all non-empty.

The first requirement can be satisfied if and only if

$$|\mathbf{Ns}| \geq 2T + 2 \tag{3.15}$$

The second requirement can always be fulfilled because $|\mathbf{B}(a)| \geq 2T + 1$ and because it is always possible to construct a T-error-correcting code with code words consisting of any number of symbols if this number of symbols is at least $2T + 1$, [MacW 78].
The third requirement is fulfilled if the general constraints of the classes $\mathcal{A}(T, 1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$ are satisfied, i.e.

$$T \geq 1 \quad \text{and} \quad b \in (\mathbf{Ns} - \{a\}) \tag{3.16}$$

and

$$(\mathbf{D} - \{a\}) \subset (\mathbf{Ns} - \{a\}) \quad \text{and} \quad |\mathbf{D} - \{a\}| \geq 2 \tag{3.17}$$

and the if and only if the condition expressed in (3.14) is satisfied, i.e.:

$$|\mathbf{Ns} - \{a\}| \geq 2 \tag{3.18}$$

Predicate (3.16) is trivially satisfied by the assumption (3.12) and the facts that $b \in \mathbf{B}(a)$ and $\mathbf{B}(a) \subset \mathbf{Ns} - \{a\}$.
Predicate (3.17) follows from (3.12) and

$$(\mathbf{D} \subset \mathbf{Ns}) \Longrightarrow ((\mathbf{D} - \{a\}) \subset (\mathbf{Ns} - \{a\}))$$

Moreover $K = 2$ and $|\mathbf{D}| \geq K + 1$ in (3.12) implies $|\mathbf{D} - \{a\}| \geq 2$.
Predicate (3.18) is implied by $K = 2$ and $|\mathbf{D}| \geq K + 1$ and $\mathbf{D} \subset \mathbf{Ns}$.

Hence for all $T, a, \mathbf{Ns}, \mathbf{D}$ with $T \geq 1$ and $a \in \mathbf{Ns}$ and $\mathbf{D} \subset \mathbf{Ns}$ and $|\mathbf{D}| \geq 3$ it holds that the class of algorithms $\mathcal{A}(T, 2, a, \mathbf{D}, \mathbf{Ns})$ is non-empty if and only if

$$|\mathbf{Ns}| \geq 2T + 2 \tag{3.19}$$

Which proves Theorem 3.1 for $K = 2$.

Suppose Theorem 3.1 holds for $K-1$ with $K \geq 3$. So suppose for all $T, a, \mathbf{Ns}, \mathbf{D}$ with $T \geq 1$ and $a \in \mathbf{Ns}$ and $\mathbf{D} \subset \mathbf{Ns}$ and $|\mathbf{D}| \geq K$ holds that the class of algorithms $\mathcal{A}(T, K-1, a, \mathbf{D}, \mathbf{Ns})$ is non-empty if and only if

$$|\mathbf{Ns}| \geq 2T + K - 1 \tag{3.20}$$

Again from our remarks (3.9) to the construction we recall that the construction of algorithms based on K rounds of information exchange is possible if and only if

1. A next-set $\mathbf{B}(a)$ can be found being a subset from $\mathbf{Ns} - \{a\}$ which contains at least $2T+1$ modules.

2. A T-error-correcting code exists of which the code words consist of $|\mathbf{B}(a)|$ symbols.

3. The classes $\mathcal{A}(T, K-1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$ of DJC algorithms with $b \in \mathbf{B}(a)$ are all non-empty.

The first requirement can be satisfied if and only if

$$|\mathbf{Ns}| \geq 2T + 2 \tag{3.21}$$

The second requirement can always be fulfilled because $|\mathbf{B}(a)| \geq 2T+1$ and because it is always possible to construct a T-error-correcting code with code words consisting of any number of symbols if this number of symbols is at least $2T+1$, [MacW 78].
The third requirement is fulfilled if the general constraints of the classes $\mathcal{A}(T, K-1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$ are satisfied, i.e.

$$T \geq 1 \text{ and } K - 1 \geq 1 \text{ and } b \in (\mathbf{Ns} - \{a\}) \tag{3.22}$$

and

$$(\mathbf{D} - \{a\}) \subset (\mathbf{Ns} - \{a\}) \text{ and } |\mathbf{D} - \{a\}| \geq K \tag{3.23}$$

and if and only if the condition expressed in (3.20) is satisfied, i.e.:

$$|\mathbf{Ns} - \{a\}| \geq 2T + K - 1 \tag{3.24}$$

Predicate (3.22) is satisfied by the assumption (3.12), the assumption $K \geq 3$, and the fact that $b \in \mathbf{B}(a)$ and $\mathbf{B}(a) \subset \mathbf{Ns} - \{a\}$.
Predicate (3.23) follows from (3.12) and

$$(\mathbf{D} \subset \mathbf{Ns}) \Longrightarrow ((\mathbf{D} - \{a\}) \subset (\mathbf{Ns} - \{a\}))$$

Moreover $|\mathbf{D}| \geq K + 1$ in (3.12) implies $|\mathbf{D} - \{b\}| \geq K$.
Predicate (3.24) is satisfied if and only if

$$|\mathbf{Ns}| \geq 2T + K \tag{3.25}$$

So the necessary and sufficient condition (3.21) for satisfying the first requirement is implied by the necessary and sufficient condition (3.25) for satisfying the third requirement. Moreover the second requirement is always satisfied.

Hence the assumption (3.20) implies that for all $T, a, \mathbf{Ns}, \mathbf{D}$ with $T \geq 1$ and $a \in \mathbf{Ns}$ and $\mathbf{D} \subset \mathbf{Ns}$ and $|\mathbf{D}| \geq K + 1$ it holds that the class of algorithms $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ is non-empty if and only if

$$|\mathbf{Ns}| \geq 2T + K \tag{3.26}$$

We already proved the theorem for $K = 2$ and thus by induction on K we obtain that (3.26) holds for any $K \geq 2$.
Which completes the proof of Theorem 3.1. □

3.3.4 Some behavioural properties of the Dispersed Joined Communication algorithms in the presence of at most T modules which behave maliciously

The behavioural properties of the Dispersed Joined Communication algorithms are expressed in the following theorem.

Theorem 3.2
Let the modules of a fully connected system be represented by the set **Ns**. *At most T of these modules behave maliciously.*
Suppose that by means of any DJC algorithm from the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ *a message* $m(a)$ *is transmitted from the source module a to the destinations represented by the set* **D**. *And let the decisions calculated in the destinations d with* $d \in \mathbf{D}$ *be denoted by* $dec_K((a), d)$, *then*

- **1.** *If the source module a and a destination d are both functioning correctly then the result $dec_K((a),d)$ of the algorithm calculated in module d equals the original message $m(a)$ in module a.*
- **2.** *For any two destinations d and e which are both functioning correctly it holds that if the results $dec_K((a),d)$ and $dec_K((a),e)$ are unequal, then the number of maliciously behaving modules in the system is at least K.* □

Proof:

We start with property 1.
Let us assume that a and d are two correctly functioning modules in the system in which a is the source module of the algorithm, $a \in \mathbf{Ns}$, and d is one of the destinations, $d \in \mathbf{D}$.
If $a = d$ then according to the construction of the algorithms $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ with $K = 1$ on page 133 and with $K > 1$ on page 135, it holds that $dec_K((a),d) = m(a)$.
So we need only to consider the case $d \neq a$, i.e. $d \in (\mathbf{D} - \{a\})$. For these cases we prove $dec_K((a),d) = m(a)$ by induction with respect to K.
Basis: $K = 1$.
From the construction we know that any algorithm in a class $\mathcal{A}(T, 1, a, \mathbf{D}, \mathbf{Ns})$ is based on only one round of information exchange. During round 0 the original message $m(a)$ in a is sent to d directly and unchanged and during round 1 the decision taken in module d equals the message received from a. So because we assume that a and d are functioning correctly it holds that $dec_1((a),d) = m(a)$.

Induction step: $K > 1$
The algorithms in a class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ have been constructed from the algorithms in the class $\mathcal{A}(T, K-1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$ with $b \in \mathbf{B}(a)$ and $\mathbf{B}(a) \subset (\mathbf{Ns} - \{a\})$.
For the latter algorithms we know from the induction hypothesis that if a message $m(a,b)$ is communicated by a correctly functioning source module b to a correctly functioning destination d in $\mathbf{D} - \{a\}$ then

$$dec_{K-1}((a,b),d) = m(a,b) \tag{3.27}$$

Moreover from the construction we know the following:
In module a by means of the encoder function $\mathcal{Y}_{(a)}$, the original message is

encoded into $|\mathbf{B}(a)|$ symbols and during round 0 each symbol is sent to a different module b, with $b \in \mathbf{B}(a)$. So

$$m(a,b) = \mathcal{Y}(a)(b)(m(a)) \tag{3.28}$$

cf. (3.10). Each of these modules b thereafter forwards the message $m(a,b)$ (is the code word symbol) to the destinations represented by the set $\mathbf{D} - \{a\}$ by means of an algorithm from the class $\mathcal{A}(T, K-1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$. As the result of these algorithms in each destination d of the set $\mathbf{D} - \{a\}$, $|\mathbf{B}(a)|$ decisions $dec_{K-1}((a,b),d)$ will become available. If module b is functioning correctly then according to (3.27) and (3.28) it holds that

$$dec_{K-1}((a,b),d) = \mathcal{Y}(a)(b)(m(a)) \tag{3.29}$$

The decoder function $\mathcal{Y}_{(a)}^{(-1)}$ is applied to these decisions $dec_{K-1}((a,b),d)$ with $b \in \mathbf{B}(a)$ in each module d. At most T modules b behave maliciously so at most T decisions do not satisfy (3.29). The code $\mathcal{Y}_{(a)}$ is T-error-correcting and thus $dec_K((a),d)$ which is the result of applying the decoder function on the values $dec_{K-1}((a,b),d)$ must be equal to $m(a)$.
This completes the proof of property 1.

Next we prove property 2.
Let us assume that two destination d and e, with $d, e \in \mathbf{D}$ behave correctly and that $dec_K((a),d) \neq dec_K((a),e)$.
If the source module a is functioning correctly then, by property 1 it holds that

$$dec_K((a),d) = m(a) = dec_K((a),e)$$

conflicting the assumption $dec_K((a),d) \neq dec_K((a),e)$. So we conclude that module a is behaving maliciously and thus $d \neq a$ and $e \neq a$.
Again we use induction with respect to K.
Basis: K=1
We already concluded from our assumption $dec_K((a),d) \neq dec_K((a),e)$ and d and e both behaving correctly, that module a is behaving maliciously. Hence the system contains at least one maliciously behaving module.

Induction step: $K > 1$
Recall that during round 0 the symbols are sent by module a to the modules b with $b \in \mathbf{B}(a)$. During the rounds $1, \cdots K$, each of these symbols $m(a,b)$

is forwarded to the destinations in the set $\mathbf{D} - \{a\}$ by means of an algorithm from the class $\mathcal{A}(T, K-1, b, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$. The results of these algorithms calculated in the modules d with $d \in (\mathbf{D}-\{a\})$ are denoted by $dec_{K-1}((a,b),d)$.

Let $dec_{K-1}((a,b),d)$ and $dec_{K-1}((a,b),e)$ with $b \in \mathbf{B}(a)$ be decisions calculated in modules d and e. We already concluded from our assumption $dec_K((a),d) \neq dec_K((a),e)$ and d and e both behaving correctly, that module a is behaving maliciously and $d \neq a$ and $e \neq a$. Hence $d, e \in (\mathbf{D}-\{a\})$. Since $d \neq a$ and $e \neq a$, the decision $dec_K((a),d)$ is based on applying the function $y_{(a)}^{(-1)}$ on the decisions $dec_{K-1}((a,b),d)$ with $b \in \mathbf{B}(a)$, whereas the decision $dec_K((a),e)$ is based on applying the same function $y_{(a)}^{(-1)}$ on the decisions $dec_{K-1}((a,b),e)$ with $b \in \mathbf{B}(a)$. It follows that

$$\forall b \; : \; b \in \mathbf{B}(a) \Longrightarrow dec_{K-1}((a,b),d) = dec_{K-1}((a,b),e) \tag{3.30}$$

would imply $dec_K((a),d) = dec_K((a),e)$. The latter however is conflicting with the assumption $dec_K((a),d) \neq dec_K((a),e)$ so we must conclude

$$\exists b \; : \; b \in \mathbf{B}(a) \wedge dec_{K-1}((a,b),d) \neq dec_{K-1}((a,b),e) \tag{3.31}$$

Recall that our assumption implies $d, e \in (\mathbf{D}-\{a\})$. So from the definition of the construction of the algorithms in the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ we know that the decisions $dec_{K-1}((a,b),d)$ are the result of the algorithms from the classes $\mathcal{A}(T, K, a, \mathbf{D}-\{a\}, \mathbf{Ns}-\{a\})$ with $b \in \mathbf{B}(a)$.
According to the induction hypothesis it holds for the latter classes that if the modules d and e are both functioning correctly and $dec_{K-1}((a,b),d) \neq dec_{K-1}((a,b),e)$ then the number of maliciously behaving modules in the set $\mathbf{Ns}-\{a\}$ must be at least $K-1$. And thus with (3.31) we conclude that the set $\mathbf{Ns}-\{a\}$ must contain at least $K-1$ maliciously behaving modules. We already concluded from the assumption that module a behaves maliciously. Hence the set $\mathbf{Ns}$ must contain at least K maliciously behaving modules. Which completes the proof of Theorem 3.2. □

3.4 A class of algorithms for reaching interactive consistency based on voting and coding

The class of algorithms for reaching interactive consistency based on voting and coding immediately follows from the Dispersed Joined Communication Algorithms defined in the previous section.

Let a system consist of N modules identified by the elements of a set **Ns**. At most T modules in this set behave maliciously. Then the classes of Interactive Consistency Algorithms among the classes of DJC algorithms are those in which the set of destinations encompasses the entire system and the number of rounds is one more than the number of maliciously behaving modules which need to be tolerated, i.e. the classes:

$$\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns}) \quad \text{with}$$
$$T \geq 1 \quad \text{and} \quad K = T + 1 \quad \text{and} \quad \mathbf{D} = \mathbf{Ns} \tag{3.32}$$

From Theorem 3.1 on page 139 we know that a class of DJC algorithms $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ with $\mathbf{D} \subset \mathbf{Ns}$ and $|\mathbf{D}| \geq K + 1$, is non-empty if and only if

$$|\mathbf{Ns}| \geq 2T + K \tag{3.33}$$

So from (3.32) and (3.33) it follows that:

Corollary 3.3 *Interactive Consistency Algorithms based on voting and coding can always be constructed if*

$$T \geq 1 \quad \textit{and} \quad N \geq 3T + 1 \quad \textit{and} \quad K = T + 1$$

The Interative Consistency Algorithms in a class $\mathcal{A}(T, T+1, a, \mathbf{Ns}, \mathbf{Ns})$ satisfy the interactive consistency requirements, as they have been defined in Definition 3.1 on page 119, i.e.:

Theorem 3.4 *Any Interactive Consistency Algorithm from the class* $\mathcal{A}(T, T+1, a, \mathbf{Ns}, \mathbf{Ns})$ *which runs on a system of* N *modules,* $N \geq 3T + 1$, *which are identified by the elements of the set* **Ns**, *of which at most* T *behave maliciously,* $T \geq 1$, *and which aims at transmitting a message* $m(a)$ *in the source*

module a to all modules of the system, satisfies the interactive consistency requirements

$$\forall d \ : \ a, d \in \overline{\mathbf{F}} \Longrightarrow dec_{T+1}((a), d) = m(a)$$

and $$\forall d, e \ : \ d, e \in \overline{\mathbf{F}} \Longrightarrow dec_{T+1}((a), d) = dec_{T+1}((a), e)$$

in which $\overline{\mathbf{F}}$ *is any set of correctly functioning modules such that*

$$|\overline{\mathbf{F}}| \geq N - T \quad \textit{and} \quad \overline{\mathbf{F}} \subset \mathbf{Ns}$$

and $dec_{T+1}((a), d)$ *with* $d \in \mathbf{Ns}$ *denotes the decision in a module d about what module a tried to send.* □

Proof:

Consider Interactive Consistency Algorithms which are defined by the non-empty class of DJC algorithms $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ with

$$T \geq 1 \ \wedge \ N \geq 3T + 1 \ \wedge \ K = T + 1 \tag{3.34}$$

Moreover let $\overline{\mathbf{F}}$ be any set of correctly functioning modules such that

$$|\overline{\mathbf{F}}| \geq N - T \quad \text{and} \quad \overline{\mathbf{F}} \subset \mathbf{Ns}$$

From Theorem 3.2 we know that

- If the source module a and a destination d are both functioning correctly then the result $dec_K((a), d)$ of the algorithm calculated in module d equals the original message $m(a)$ in module a.

Thus:

$$\forall d \ : \ a, d \in \overline{\mathbf{F}} \Longrightarrow dec_K((a), d) = m(a) \tag{3.35}$$

Which proves the first part of the interactive consistency property.

From the second part of Theorem 3.2 we know that

- For any two destinations d and e which are both functioning correctly it holds that if the results $dec_K((a), d)$ and $dec_K((a), e)$ are unequal, then the number of maliciously behaving modules is at least K.

However this conflicts with the constraint $K = T + 1$ and the assumption that at most T modules behave maliciously. Thus if both modules d and e are behaving correctly, the decisions $dec_K((a), d)$ and $dec_K((a), e)$ must be identical. So

$$\forall d, e \; : \; d, e \in \overline{\mathbf{F}} \Longrightarrow dec_K((a), d) = dec_K((a), e) \tag{3.36}$$

Which completes the proof of Theorem 3.4. □

3.5 Some remarks on the construction of Interactive Consistency Algorithms which are based on voting and coding

In the previous sections we have defined the Interactive Consistency Algorithms which are based on voting and coding, starting from the Dispersed Joined Communication algorithms. In this section we will first discuss the design process and the design freedom which is left by the definition of these IAC algorithms. Thereafter we will elucidate the design of the Interactive Consistency Algorithms based on voting and coding with two examples.
In the introduction to this chapter we claimed that the reduction of the number of messages which needs to be transmitted between the modules can be obtained in two ways, i.e.:

- by minimizing the number of directions in which the messages are broadcast each round, and
- by replacing the voting function by an error-correcting code.

From the construction we immediately see that replacing the voting function by an error-correcting code causes an increase in the number of modules to which a modified message has to be sent. However, the size of the messages to be transmitted decreases compared to the original message in the source. In the next chapter we will show that choosing an error-correcting code is more efficient than reducing the number of directions in which a message is sent each round. The advantage of fewer messages to be transmitted due to the application of non-trivial error-correcting, has to be paid for by a larger minimal size of the original message in the source and by the fact that the implementation of the decoding function of an error-correcting code is much more complex than the implementation of a simple majority voter.

For these reasons we will define in Section 3.5.3 two subclasses of algorithms, i.e

- The *Minimal Voting* algorithms. In this class of algorithms the number of directions in which a message is sent in each round, is minimized and in the decision-making process only majority voting is applied.
- The *Maximal Coding* algorithms. In this class of algorithms the messages are broadcast to as many modules as is allowed by the definition of DJC algorithms, such that the amount of redundancy in an error-correcting code word is as little as possible and the size reduction of the messages in each step is maximal.

Another way of minimizing the amount of messages is to reduce the amount of destinations in which the decisions are calculated and thereafter to broadcast the results in an additional round to the other modules. This *Subset Method* will be the issue of Section 3.5.4.

3.5.1 The general construction of Interactive Consistency Algorithms which are based on voting and coding

Suppose we want to design an IAC algorithm for a system consisting of N modules of which at most T may behave maliciously. Such a construction is always possible if

$$T \geq 1 \ \wedge \ N \geq 3T+1 \ \wedge \ K = T+1$$

cf. Corollary 3.3.

The broadcasting process

From the definition of the IAC algorithms on voting and coding we know that the source module a communicates the original message $m(a)$ by means of a DJC algorithm from the class $\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns})$ with $T = K + 1$ and $\mathbf{D} = \mathbf{Ns}$ to all modules in $\mathbf{D}$. The source module in this case is always one of the destinations and thus the original message $m(a)$ is kept stored in the source module. (cf. item 1.a of the definition of the construction on page 135). The decision $dec_K((a), a)$ in the source equals the original message $m(a)$, which has been kept stored in the source until round K, (cf. 3 of the definition of the construction).

The source module a communicates the original message $m(a)$ to the other modules, i.e the modules in $\mathbf{Ns} - \{a\}$, as follows:
Module a sends during round 0 (modified) messages $m(a,b)$ to a number of modules identified by the next-set $\mathbf{B}(a)$. Thus $b \in \mathbf{B}(a)$. Notice that $\mathbf{B}(a) \subset (\mathbf{Ns} - \{a\})$. and thus the number of elements in $\mathbf{B}(a)$ is at most $N-1$. The modified messages which are forwarded by module a are partially encoded copies of the original message. The choice of the T-error-correcting code $y_{(a)}$ which is used in module a for encoding is limited by the number of modules to which symbols can be sent, i.e., the number $n_{(a)}$ of symbols of a code word of the T-error-correcting code equals $|\mathbf{B}(a)|$ and thus may be at most $N-1$. Furthermore T-error-correcting codes consisting of code words of $n_{(a)}$ symbols exist if and only if $n_{(a)} \geq 2T+1$, provided the symbol size $b_{(a)}$ is sufficient large. So

$$2T+1 \leq n_{(a)} \leq N-1$$

Obviously for $T=1$ and $N=4$, i.e. the most simple IAC algorithm, no choice is left, but in all other cases a code can be freely chosen within the preceding constraint.

From item 2 of the definition of the construction on page 135 we know that each message $m(a,b)$ received by a modules b, with $b \in \mathbf{B}(a)$ is communicated to the destinations d in the set $\mathbf{D} - \{a\}$, by means of an algorithm from the class $\mathcal{A}(T, K-1, b, \mathbf{D} - \{a\}, \mathbf{Ns} - \{a\})$, in which $K-1 = T$ and $(\mathbf{D} - \{a\}) = (\mathbf{Ns} - \{a\})$.

The algorithms from the class $\mathcal{A}(T, T, b, \mathbf{Ns} - \{a\}, \mathbf{Ns} - \{a\})$, with $b \in \mathbf{B}(a)$ will again after encoding, forward the message $m(a,b)$ to the modules c with $c \in \mathbf{B}(a,b)$. In which $\mathbf{B}(a,b)$ is the next-set which belongs to the algorithm chosen from the class $\mathcal{A}(T, T, b, \mathbf{Ns} - \{a\}, \mathbf{Ns} - \{a\})$.
Clearly the module b will be one of the destinations, so due to the algorithm chosen the message $m(a,b)$ will also be kept stored in module b and become the decision $dec_{K-1}((a,b),b)$ during round K.
The messages mentioned before, which are received during round 1 by the modules c, with $c \in \mathbf{B}(a,b)$, are denoted by $m(a,b,c)$.
These messages $m(a,b,c)$ are again communicated to the modules d in the set $\mathbf{Ns} - \{a,b\}$ by means of an algorithm from the class $\mathcal{A}(T, T-1, c, \mathbf{Ns} - \{a,b\}, \mathbf{Ns} - \{a,b\})$ and because $c \in \mathbf{B}(a,b)$ and $\mathbf{B}(a,b) \subset \mathbf{Ns} - \{a,b\}$ the module c also will be one of the destinations and therefore the message

$m(a,b,c)$ will also be kept stored.

Notice that in a particular module c more than one message might arrive. For example, suppose that the messages $m(a,b_1,c)$ and $m(a,b_2,c)$ arrive in module c. Thus $b_1,b_2 \in \mathbf{B}(a)$, and $c \in \mathbf{B}(a,b_1)$, and $c \in \mathbf{B}(a,b_2)$. In that case of course the choice of the algorithm from the classes $\mathcal{A}(T,T-1,c,\mathbf{Ns}-\{a,b_1\},\mathbf{Ns}-\{a,b_1\})$ and $\mathcal{A}(T,T-1,c,\mathbf{Ns}-\{a,b_2\},\mathbf{Ns}-\{a,b_2\})$ for forwarding the messages $m(a,b_1,c)$ and $m(a,b_2,c)$ may be made independently. This process is continued until the last round of information exchange, i.e. round $K-1$. During this round each message $m(a,\underline{s},p)$ which arrives in a module p, is forwarded to the destinations in the set $\mathbf{Ns}-\text{set}(a,\underline{s})$ by means of the only algorithm in the class $\mathcal{A}(T,1,p,\mathbf{Ns}-\text{set}(a,\underline{s}),\mathbf{Ns}-\text{set}(a,\underline{s}))$. In which, $\text{set}(a,\underline{s})$ denotes the set of modules from which the string $(a,\underline{s})$ is composed.

From the preceding and the definition of the construction on page 135 it follows that:
A particular algorithm in the class $\mathcal{A}(T,K,a,\mathbf{D},\mathbf{Ns})$ is fully determined by

- the choice of the set $\mathbf{B}(a)$,
- the choice of the code $\mathcal{Y}_{(a)}$, and
- for each module b with $b \in \mathbf{B}(a)$, the choice of the algorithm in the class $\mathcal{A}(T,K-1,b,\mathbf{D}-\{a\},\mathbf{Ns}-\{a\})$ by means of which the message $m(a,b)$ is forwarded to the destinations in $\mathbf{D}-\{a\}$.

Similarly a particular algorithm in the class $\mathcal{A}(T,K-t,p,\mathbf{D}-\text{set}(a,\underline{s}),\mathbf{Ns}-\text{set}(a,\underline{s}))$, with $|(a,\underline{s})| = t$, which is utilized by an algorithm from the class $\mathcal{A}(T,K,a,\mathbf{D},\mathbf{Ns})$ for forwarding a message $m(a,\underline{s},p)$, is fully determined by

- the choice of the set $\mathbf{B}(a,\underline{s},p)$,
- the choice of the code $\mathcal{Y}_{(a,\underline{s},p)}$, and
- for each module q with $q \in \mathbf{B}(a,\underline{s},p)$, the choice of the algorithm in the class $\mathcal{A}(T,K-t-1,q,\mathbf{D}-\text{set}(a,\underline{s},p),\mathbf{Ns}-\text{set}(a,\underline{s},p))$ by means of which the message $m(a,\underline{s},p,q)$ is forwarded to the destinations in $\mathbf{D}-\text{set}(a,\underline{s},p)$.

Notice again that for each message $m(a, \underline{s}, p)$ the choice of the algorithm from the class $\mathcal{A}(T, K-t, p, \mathbf{D}-\text{set}(a,\underline{s}), \mathbf{Ns}-\text{set}(a,\underline{s}))$ may be made independently and only is restricted by the definition of the construction.

Obviously, not every string of length $K+1$ or less over the set **Ns** will identify a message caused by a particular IAC algorithm.
Therefore, let **Sm** be the set of all strings which identify messages which are caused by a particular IAC algorithm from the class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$, then **Sm** will be determined by

$$(a) \in \mathbf{Sm}$$
$$\left((\underline{s}) \in \mathbf{Sm} \wedge p \in \mathbf{B}(\underline{s}) \right) \Longrightarrow (\underline{s}, p) \in \mathbf{Sm} \tag{3.37}$$

Clearly the sets $\mathbf{B}(\underline{s})$, are only defined for $1 \leq |\underline{s}| \leq K$ and $\underline{s} \in \mathbf{Sm}$.

Recall that the set $\mathbf{B}(\underline{s}, p)$ amongst other things determines a particular algorithm in the class $\mathcal{A}(T, K-t, p, \mathbf{D}-\text{set}(\underline{s}), \mathbf{Ns}-\text{set}(\underline{s}))$, with $|\underline{s}| = t$, which is utilized by an algorithm from the class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ for forwarding a message $m(\underline{s}, p)$.
This set $\mathbf{B}(\underline{s}, p)$ will have to satisfy the rules imposed by the definition of the construction on page 135, i.e.:

$$\begin{aligned} \left((\underline{s}, p) \in \mathbf{Sm} \wedge 1 \leq |(\underline{s}, p)| \leq K-1 \right) \Longrightarrow \\ \Big(\left(\mathbf{B}(\underline{s}, p) \subset (\mathbf{Ns} - \text{set}(\underline{s}, p)) \right) \\ \wedge \left(2T+1 \leq |\mathbf{B}(\underline{s}, p)| \leq |\mathbf{Ns} - \text{set}(\underline{s}, p)| \right) \Big) \end{aligned} \tag{3.38}$$

The last round of information exchange during the broadcasting process is determined by the algorithms from the classes $\mathcal{A}(T, 1, p, \mathbf{Ns}-\text{set}(\underline{s}), \mathbf{Ns}-\text{set}(\underline{s}))$, with $|\underline{s}| = K-1$. These algorithms forward the messages directly to the destinations. So for these algorithms the next-set is defined by

$$\left((\underline{s}, p) \in \mathbf{Sm} \wedge |(\underline{s}, p)| = K \right) \Longrightarrow \mathbf{B}(\underline{s}, p) = (\mathbf{Ns} - \text{set}(\underline{s}, p)) \tag{3.39}$$

From the preceding it therefore follows that a particular algorithm in the class
$\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ is fully determined by the sets $\mathbf{B}(\underline{s})$ with $\underline{s} \in \mathbf{Sm}$ and the T-error-correcting codes $y_{(\underline{s})}$ with $\underline{s} \in \mathbf{Sm}$ and $1 \leq |\underline{s}| \leq K-1$.

The decision-making process

From the definition of the construction on page 135, of the DJC algorithms in a class $\mathcal{A}(T, K-t, p, \mathbf{Ns}-\text{set}(\underline{\mathbf{s}}), \mathbf{Ns}-\text{set}(\underline{\mathbf{s}}))$, with $|\underline{\mathbf{s}}| = t$, we know that the decision-making process of such an algorithm which is executed in the "source module" p, only consists of taking the message $m(\underline{\mathbf{s}}, p)$ which had been kept stored since round t. Thus

$$dec_{K-t}((\underline{\mathbf{s}}, p), p) = m(\underline{\mathbf{s}}, p) \tag{3.40}$$

Moreover we know that the decision-making process of such an algorithm which is executed in a module d, with $d \in (\mathbf{Ns}-\text{set}(\underline{\mathbf{s}}, p))$ encompasses the $n_{(\underline{\mathbf{s}},p)}$ decision-making processes of the algorithms chosen from the class $\mathcal{A}(T, K-t-1, q, \mathbf{Ns}-\text{set}(\underline{\mathbf{s}}, p), \mathbf{Ns}-\text{set}(\underline{\mathbf{s}}, p))$, with $q \in \mathbf{B}(\underline{\mathbf{s}}, p)$ followed by the execution of the decoder function $\mathcal{Y}^{(-1)}_{(\underline{\mathbf{s}},p)}$, cf. Figures 3.5 and 3.6 on page 136.

So the decision-making process executed by an algorithm from the class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$, in a module d during round K starts with the calculation of the results $dec_1((\underline{\mathbf{u}}, p, q), d)$ of the algorithms chosen from the classes $\mathcal{A}(T, 1, q, \mathbf{Ns}-\text{set}(\underline{\mathbf{u}}, p), \mathbf{Ns}-\text{set}(\underline{\mathbf{u}}, p))$, which forwarded the messages $m(\underline{\mathbf{u}}, p, q)$ to module d in the set $\mathbf{D}-\text{set}(\underline{\mathbf{u}}, p)$. Obviously $(\underline{\mathbf{u}}, p, q) \in \mathbf{Sm}$ and any string $\underline{\mathbf{u}}$ starts with a.
The decision $dec_1((\underline{\mathbf{u}}, p, q), d)$ with $p \neq d$ equals the message $m(\underline{\mathbf{u}}, p, q, d)$ received by module d during round $K-1$. Because $\mathbf{D} = \mathbf{Ns}$ and the message $m(\underline{\mathbf{u}}, p, q)$ is communicated to the destinations in $\mathbf{D}-\text{set}(\underline{\mathbf{u}}, p)$ it holds that q is one of the destinations. So $d \in \mathbf{B}(\underline{\mathbf{u}}, p)$. And according to the construction $dec_1((\underline{\mathbf{u}}, p, d), d)$ equals the message $m(\underline{\mathbf{u}}, p, d)$ received by module d during round $K-2$.
So for all destinations d with $d \in (\mathbf{Ns}-\text{set}(\underline{\mathbf{u}}, p))$ it holds that:

$$\begin{aligned} q \neq d &\Longrightarrow dec_1((\underline{\mathbf{u}}, p, q), d) = m(\underline{\mathbf{u}}, p, q, d) \\ & dec_1((\underline{\mathbf{u}}, p, d), d) = m(\underline{\mathbf{u}}, p, d) \end{aligned} \tag{3.41}$$

So in all destinations d represented by $\mathbf{D}-\text{set}(\underline{\mathbf{u}}, p)$, the decisions $dec_1((\underline{\mathbf{u}}, p, q), d)$ can be calculated.

After the calculation of the decisions $dec_1((\underline{\mathbf{u}}, p, q), d)$ in the modules d in $\mathbf{Ns}-\text{set}(\underline{\mathbf{u}}, p)$, the decisions $dec_2((\underline{\mathbf{u}}, p), d)$ of the algorithms chosen from the

classes $\mathcal{A}(T, 2, q, \mathbf{Ns}-\text{set}(\underline{\mathbf{u}}), \mathbf{Ns}-\text{set}(\underline{\mathbf{u}}))$, which communicated the messages $m(\underline{\mathbf{u}}, p)$ to modules d in $\mathbf{Ns} - \text{set}(\underline{\mathbf{u}})$, are calculated in these modules d from $\mathbf{Ns} - \text{set}(\underline{\mathbf{u}})$.
If $d = p$ the decision $dec_2((\underline{\mathbf{u}}, p), d)$ equals the message $m(\underline{\mathbf{u}}, p)$ which was kept stored in module p. So what remains to be shown is the calculation of the decisions in the d in $\mathbf{Ns} - \text{set}(\underline{\mathbf{u}}, p)$.
The modules q to which the message $m(\underline{\mathbf{u}}, p)$ is sent after encoding are determined by the set $\mathbf{B}(\underline{\mathbf{u}}, p)$. We already concluded that in all modules d with $d \in (\mathbf{Ns} - \text{set}(\underline{\mathbf{u}}, p))$ decisions $dec_1((\underline{\mathbf{u}}, p, q), d)$ have been calculated first.
So in these modules d for each message $m(\underline{\mathbf{u}}, p)$ received by module p a number of $n_{(\underline{\mathbf{u}},p)}$ decisions $dec_1((\underline{\mathbf{u}}, p, q), d)$ are available with $q \in \mathbf{B}(\underline{\mathbf{u}}, p)$. the decoder function $y^{(-1)}_{(\underline{\mathbf{u}},p)}$ is applied to these decisions, which results in the decisions $dec_2((\underline{\mathbf{u}}, p), d)$.

The decisions $dec_2((\underline{\mathbf{u}}, p), d)$ which have been calculated thus far in the modules d in $\mathbf{Ns} - \text{set}(\underline{\mathbf{u}})$, could be regarded as estimates calculated by the modules d of the message value $m(\underline{\mathbf{u}}, p)$ received by module p during round $K-3$.

The first round of applying decoder functions $y^{(-1)}_{(\underline{\mathbf{u}},p)}$, together with $dec_2((\underline{\mathbf{u}}, d), d)$, delivers for each $\underline{\mathbf{u}}$ a number of $n_{(\underline{\mathbf{u}})}$ decisions $dec_2((\underline{\mathbf{u}}, p), d)$ with $p \in \mathbf{B}(\underline{\mathbf{u}})$ and $d \in (\mathbf{Ns} - \text{set}(\underline{\mathbf{u}}))$. The decoder function $y^{(-1)}_{(\underline{\mathbf{u}})}$ is applied to these decisions, which results in the decisions $dec_3((\underline{\mathbf{u}}), d)$.

This process is continued until $dec_K((a), d)$ has been calculated by means of the decoder function $y^{(-1)}_{(a)}$ for $d \neq a$ and $dec_K((a), a)$ is obtained from $m(a)$.

3.5.2 Two simple examples

In order to elucidate the general construction method of Intractive Consistency Algorithms based on voting and coding we will present two examples in detail.
The first example is the most simple algorithm, i.e. $T = 1$ and $N = 4$, thus an algorithm from the class $\mathcal{A}(1, 2, 0, \{0, 1, 2, 3\}, \{0, 1, 2, 3\})$. We will see that in this class there is only one algorithm and that this algorithm is identical to the Pease algorithm.

$N = 4$ T=1 $K = 2$	
$\mathbf{Ns} = \{0, 1, 2, 3\}$	
$\mathbf{B}(0) = \{1, 2, 3\}$	$\mathbf{B}(0, 1) = \{2, 3\}$ $\mathbf{B}(0, 2) = \{1, 3\}$ $\mathbf{B}(0, 3) = \{1, 2\}$

Table 3.1: *The next-sets*

$N = 4$ T=1 $K = 2$				
m(0)	$\Longrightarrow$	$m(0, 1)$	$\Longrightarrow$	$m(0, 1, 2)$ $m(0, 1, 3)$
		$m(0, 2)$	$\Longrightarrow$	$m(0, 2, 1)$ $m(0, 2, 3)$
		$m(0, 3)$	$\Longrightarrow$	$m(0, 3, 1)$ $m(0, 3, 2)$

Table 3.2: *The messages*

$N = 4 \quad T{=}1 \quad K = 2$

$m(0)$			$\Longrightarrow$	$dec_2((0),0)$
$m(0,1)$	$\Longrightarrow$	$dec_1((0,1),1)$		
$m(0,2,1)$	$\Longrightarrow$	$dec_1((0,2),1)$	$\Longrightarrow$	$dec_2((0),1)$
$m(0,3,1)$	$\Longrightarrow$	$dec_1((0,3),1)$		
$m(0,1,2)$	$\Longrightarrow$	$dec_1((0,1),2)$		
$m(0,2)$	$\Longrightarrow$	$dec_1((0,2),2)$	$\Longrightarrow$	$dec_2((0),2)$
$m(0,3,2)$	$\Longrightarrow$	$dec_1((0,3),2)$		
$m(0,1,3)$	$\Longrightarrow$	$dec_1((0,1),3)$		
$m(0,2,3)$	$\Longrightarrow$	$dec_1((0,2),3)$	$\Longrightarrow$	$dec_2((0),3)$
$m(0,3)$	$\Longrightarrow$	$dec_1((0,3),3)$		

Table 3.3: *The decision-making process*

The sets $\mathbf{B}(\underline{\mathbf{s}})$ are shown in Table 3.1. The source has to broadcast its message to a number of modules which is determined by the code. However, remember that in this example no choice is left and the only possibility is a $(3,1)$ repetition code. Thus module 0 sends the original message unchanged to 3 different modules, i.e. the modules in the set $\{1,2,3\}$. These modules a forward the message $m(0,a)$ by means of the only algorithm in the class $\mathcal{A}(1,1,a,\mathbf{Ns}-\{a\},\mathbf{Ns}-\{a\})$ with $\mathbf{Ns} = \{0,1,2,3\}$.

The sets $\mathbf{Sm}$ and $\mathbf{B}(0,a)$ can be easily calculated by means of the relations (3.37), (3.38), and (3.39). It is easily seen that in this example there is no design freedom, except in the naming of the modules.

The messages caused by the algorithm, thus the messages $m(\underline{\mathbf{s}})$ with $\underline{\mathbf{s}} \in \mathbf{Sm}$ are shown in Table 3.2. Notice that a message $m(0,1)$ represents the message received at the end of round 0 by module 1 from module 0. The data dependency between the messages is indicated in the figure by $\Longrightarrow|$. For example $m(0,1) \Longrightarrow |m(0,1,2)$ indicates that the message $m(0,1,2)$ received by module 2 from module 1 is obtained from $m(0,1)$ by applying a partial encoding function on $m(0,1)$. In this case the partial encoder function is the identity function.

The flattened (= non-recursive) representation of the decision-making process is given in Table 3.3. The decision-making process in a module is based on all messages received by the module during the entire broadcast process. From the preceding we know that the decision-making process starts with the calculation of the decisions $dec_1((0,a),d)$. The messages $m(0,a)$ are forwarded to the destinations during round 1 by means of an algorithm from the class $\mathcal{A}(1,1,a,\{1,2,3\},\{1,2,3\})$ with $a \in \mathbf{B}(0)$. So according to the definition of the construction of the algorithms with $K = 1$ on page 132, it holds that $dec_1((0,1),1) = m(0,1)$, $dec_1((0,2),1) = m(0,2,1)$, and $dec_1((0,3),1) = m(0,3,1)$. In the second step of the decision-making process the decisions $dec_2((0),d)$ are calculated, i.e. $dec_2((0),0) = m(0)$ and $dec_2((0),1)$ is obtained by taking the majority vote over the values $dec_1((0,1),1)$, $dec_1((0,2),1)$, and $dec_1((0,3),1)$. The decisions $dec_2((0),2)$ and $dec_2((0),3)$ are obtained similarly.

The way in which the decisions $dec_{K-t}((\underline{s}),a)$ are calculated from the decisions $dec_{K-t-1}((\underline{s},b),a)$ with $b \in \mathbf{B}_t(\underline{s})$, by means of the decoder function $\gamma_{(\underline{s})}^{(-1)}$ is denoted in the table by $| \Longrightarrow$.

The second example is a little more complex and concerns an algorithm from the class $\mathcal{A}(2,3,0,\mathbf{Ns},\mathbf{Ns})$ with $\mathbf{Ns} = \{0,1,\cdots,6\}$. Thus $T = 2$, $K = 3$, and $N = 7$. The sets, the messages and the decision-making process are given in the Tables 3.4, 3.5, and 3.6. This example again meets the $3T+1$ bound and the $K+1$ bound. The example provides some more design freedom.

The next-set $\mathbf{B}(0)$ should be contained in the set $\{1,\cdots,6\}$, cf. (3.38). Moreover this next-set must contain at least $2T+1 = 5$ modules. Hence we may choose between 5 and 6 modules. The first choice would again imply a repetition code and therefore we choose $\mathbf{B}(0) = \{1,\cdots,6\}$.
Bear in mind that T-error-correcting codes can always be constructed with $n_{(0)} = k_{(0)} + 2T$ and any symbol size of $b_{(0)} \geq \log_2(n_{(0)} - 1)$ bits in which $n_{(0)}$ is the number of symbols of a code word and $k_{(0)}$ is the number of symbols of a data word. Hence a code with $n_{(0)} = 6$, $k_{(0)} = 2$, and $b_{(0)} = 3$ suffices. From the preceding it follows that the minimum size *msize* of the original message in the source is 6, i.e. 2 symbols of size 3.

During round 0, in the source module 0, the original message is encoded into 6 symbols of 3 bits. Each of these symbols is sent to a different module

$N = 7$ T=2 $K = 3$

$\mathbf{Ns} = \{0, 1, 2, 3, 4, 5, 6\}$

$B(0) = \{1,2,3,4,5,6\}$	$B(0,1) = \{2,3,4,5,6\}$	$B(0,1,2) = \{3,4,5,6\}$ $B(0,1,3) = \{2,4,5,6\}$ $B(0,1,4) = \{2,3,5,6\}$ $B(0,1,5) = \{2,3,4,6\}$ $B(0,1,6) = \{2,3,4,5\}$
	$B(0,2) = \{1,3,4,5,6\}$	.....
	$B(0,3) = \{1,2,4,5,6\}$	.....
	$B(0,4) = \{1,2,3,5,6\}$	.....
	$B(0,5) = \{1,2,3,4,6\}$	.....
	$B(0,6) = \{1,2,3,4,5\}$	$B(0,6,1) = \{2,3,4,5\}$ $B(0,6,2) = \{1,3,4,5\}$ $B(0,6,3) = \{1,2,4,5\}$ $B(0,6,4) = \{1,2,3,5\}$ $B(0,6,5) = \{1,2,3,4\}$

Table 3.4: *The next-sets*

$N = 7$ T=2 $K = 3$

				$m(0,1,2)$	$\Longrightarrow$	$m(0,1,2,3)$ $m(0,1,2,4)$ $m(0,1,2,5)$ $m(0,1,2,6)$
				$m(0,1,3)$	$\Longrightarrow$	$m(0,1,3,2)$ $m(0,1,3,4)$ $m(0,1,3,5)$ $m(0,1,3,6)$
		$m(0,1)$	$\Longrightarrow$	$m(0,1,4)$	$\Longrightarrow$	.....
				$m(0,1,5)$	$\Longrightarrow$	.....
				$m(0,1,6)$	$\Longrightarrow$	$m(0,1,6,2)$ $m(0,1,6,3)$ $m(0,1,6,4)$ $m(0,1,6,5)$
		$m(0,2)$	$\Longrightarrow$	.....		
		$m(0,3)$	$\Longrightarrow$	.....		
m(0)	$\Longrightarrow$	$m(0,4)$	$\Longrightarrow$	.....		
		$m(0,5)$	$\Longrightarrow$	.....		
				$m(0,6,1)$	$\Longrightarrow$	$m(0,6,1,2)$ $m(0,6,1,3)$ $m(0,6,1,4)$ $m(0,6,1,5)$
				$m(0,6,2)$	$\Longrightarrow$	.....
		$m(0,6)$	$\Longrightarrow$	$m(0,6,3)$	$\Longrightarrow$	.....
				$m(0,6,4)$	$\Longrightarrow$	.....
				$m(0,6,5)$	$\Longrightarrow$	$m(0,6,5,1)$ $m(0,6,5,2)$ $m(0,6,5,3)$ $m(0,6,5,4)$

Table 3.5: *The messages*

$N = 7 \quad T=2 \quad K = 3$

$m(0)$					$\Longrightarrow$	$dec_3((0),0)$
$m(0,1)$			$\Longrightarrow$	$dec_2((0,1),1)$		
$m(0,2,1)$	$\Longrightarrow$	$dec_1((0,2,1),1)$				
$m(0,2,3,1)$	$\Longrightarrow$	$dec_1((0,2,3),1)$				
$m(0,2,4,1)$	$\Longrightarrow$	$dec_1((0,2,4),1)$	$\Longrightarrow$	$dec_2((0,2),1)$		
$m(0,2,5,1)$	$\Longrightarrow$	$dec_1((0,2,5),1)$				
$m(0,2,6,1)$	$\Longrightarrow$	$dec_1((0,2,6),1)$				
$m(0,3,1)$	$\Longrightarrow$	$dec_1((0,3,1),1)$				
$m(0,3,2,1)$	$\Longrightarrow$	$dec_1((0,3,2),1)$				
$m(0,3,4,1)$	$\Longrightarrow$	$dec_1((0,3,4),1)$	$\Longrightarrow$	$dec_2((0,3),1)$		
$m(0,3,5,1)$	$\Longrightarrow$	$dec_1((0,3,5),1)$				
$m(0,3,6,1)$	$\Longrightarrow$	$dec_1((0,3,6),1)$				
		.....	$\Longrightarrow$	$dec_2((0,4),1)$	$\Longrightarrow$	$dec_3((0),1)$
		.....	$\Longrightarrow$	$dec_2((0,5),1)$		
$m(0,6,1)$	$\Longrightarrow$	$dec_1((0,6,1),1)$				
$m(0,6,2,1)$	$\Longrightarrow$	$dec_1((0,6,2),1)$				
$m(0,6,3,1)$	$\Longrightarrow$	$dec_1((0,6,3),1)$	$\Longrightarrow$	$dec_2((0,6),1)$		
$m(0,6,4,1)$	$\Longrightarrow$	$dec_1((0,6,4),1)$				
$m(0,6,5,1)$	$\Longrightarrow$	$dec_1((0,6,5),1)$				
				.....	$\Longrightarrow$	$dec_3((0),2)$
				.....	$\Longrightarrow$	$dec_3((0),3)$
				.....	$\Longrightarrow$	$dec_3((0),4)$
				.....	$\Longrightarrow$	$dec_3((0),5)$
				.....	$\Longrightarrow$	$dec_3((0),6)$

Table 3.6: *The decision-making process*

of the set $\mathbf{B}(0)$. Thereafter during the rounds 1 and 2 these symbols are forwarded to the destinations by means of the algorithms from the classes $\mathcal{A}(2, 2, a, \mathbf{Ns} - \{0\}, \mathbf{Ns} - \{0\})$. The next-sets $\mathbf{B}(0, a)$ can be easily derived by means of the relation (3.38), i.e.

$$\mathbf{B}(0, a) \subset (\mathbf{Ns} - \{0, a\})$$

$$2T + 1 \leq |\mathbf{B}(0, a)| \leq |\mathbf{Ns} - \{0, a\}|$$

Because $|\mathbf{Ns}| = 7$ and $T = 2$ the only choice left for $\mathbf{B}(0, a)$ is $\mathbf{Ns} - \{0, a\}$ and the only choice left for the code applied by the algorithms is a repetition code.

The dependency between the messages is illustrated in Table 3.5 and the decision-making process is illustrated in Table 3.6, both in the same way as in the previous example.

3.5.3 The Minimal Voting algorithms and the Maximal Coding algorithms

The Minimal Voting algorithm

For each class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ of Interactive Consistency Algorithms which are based on voting and coding, a subclass of algorithms exists which is entirely based on voting. In this subclass thus only repetition codes are applied and consequently during each round of the broadcast process a received message $m(\underline{s})$ is broadcast unchanged to a number of modules given by the set $\mathbf{B}(\underline{s})$. In order to reduce the amount of transmitted information as much as possible, during each round except the last one, each message is relayed to exactly $2T + 1$ modules, which is the minimum number that is required, cf. (3.38). During the last round a message $m(\underline{s}, p)$ is sent to the modules in $\mathbf{Ns} - \mathrm{set}(\underline{s}, p)$, cf. (3.39).
Algorithms which satisfy the preceding requirements are called *Minimal Voting algorithms.*

So the subclass of Minimal Voting algorithms within a class of algorithms $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ with

$$N \geq 3T + 1 \quad \text{and} \quad K = T + 1$$

is made up of those algorithms which satisfy

$$|\mathbf{B}(\underline{s})| = 2T + 1 \quad \text{for} \quad \underline{s} \in \mathbf{Sm} \quad \text{and} \quad 1 \leq |\underline{s}| \leq K - 1 \tag{3.42}$$

The Maximal Coding algorithm

For each class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ of Interactive Consistency Algorithms which is based on voting and coding, another subclass of algorithms exists in which the advantages of using error-correcting codes are used to the full in order to reduce the amount of information which needs to be transmitted. The algorithms in this subclass are called *Maximal Coding algorithms.*

When a T-error-correcting code $\mathcal{Y}_{(\underline{s})}$ is applied consisting of code words of $n_{(\underline{s})}$ symbols of size $b_{(\underline{s})}$ and data words consisting of $k_{(\underline{s})}$ symbols of the same size, then the total amount of information broadcast by a module during round t is a factor $n_{(\underline{s})}/k_{(\underline{s})}$ more than the amount of information received by that module during round $t-1$.
From [MacW 78] we know that a T-error-correcting code $\mathcal{Y}_{(\underline{s})}$ can be constructed if and only if $n_{(\underline{s})} \geq k_{(\underline{s})} + 2T$.

In the subclass of Maximal Coding algorithms, the fraction $n_{(\underline{s})}/k_{(\underline{s})}$ is kept as low as possible. So the partially encoded messages are sent to as many modules as is allowed by the construction of the DJC algorithms and we choose $n_{(\underline{s})} = k_{(\underline{s})} + 2T$. Codes with these parameters always exist if the symbol size $b_{(\underline{s})}$ satisfies

$$b_{(\underline{s})} \geq \log_2(n_{(\underline{s})} - 1)$$

Recall requirement (3.38), i.e.

$$2T + 1 \leq |\mathbf{B}(\underline{s})| \leq |\mathbf{Ns} - \mathrm{set}(\underline{s})| \quad \text{for} \quad 1 \leq |\underline{s}| \leq K - 1 \quad \text{and} \quad \underline{s} \in \mathbf{Sm}$$

We choose the set $\mathbf{B}(\underline{s})$ to be as large as possible, thus

$$|\mathbf{B}(\underline{s})| = |\mathbf{Ns} - \mathrm{set}(\underline{s})| \quad \text{for} \quad 1 \leq |\underline{s}| \leq K - 1 \quad \text{and} \quad \underline{s} \in \mathbf{Sm} \tag{3.43}$$

Let t denote the round in which the code $\mathcal{Y}_{(\underline{s})}$ is applied. So $|\underline{s}| = t + 1$.
From (3.43) then follows

$$n_{(\underline{s})} = |\mathbf{B}(\underline{s})| = N - t - 1 \quad \text{for} \quad 0 \leq t \leq K - 2 \tag{3.44}$$

So the subclass of Maximal Coding algorithms within class $\mathcal{A}(T, K, a, \mathbf{Ns}, \mathbf{Ns})$ of Interactive Consistency Algorithms which are based on voting and coding, is characterized by the parameters of the applied T-error-correcting codes as follows:
For each round t with $0 \leq t \leq K-2$ a T-error-correcting code is applied with parameters

number of code word symbols: $n_{(\underline{s},a)} = N - t - 1$

number of data word symbols: $k_{(\underline{s},a)} = N - t - 1 - 2T$

symbol size in number of bits:

if $N - t - 1 \geq 2T + 1$ then: $b_{(\underline{s},a)} \geq \log_2(N - t - 2)$

if $N - t - 1 = 2T + 1$ then: $b_{(\underline{s},a)} \geq 1$

(3.45)

Notice that in the case of a repetition code, i.e. $N - t - 1 = 2T + 1$, the minimal symbol size is only 1.
Because during each round of information exchange, except the last, a partial encoder function is applied to the message, the message size decreases each round. We will elaborate on this in the next chapter.
If $N = 3T+1$ the code used during round $K-2$ is always a simple repetition code. This follows immediately from (3.45) and $K = T + 1$. Notice again that during the last round, i.e. round $K - 1$, no code is applied and the messages are forwarded unchanged to their destinations.

3.5.4 The Subset Method

In Chapter 2, it has been shown that once agreement among N' modules, with $N' \geq 2T + 1$ in a set of N modules of which at most T are faulty, is obtained, the agreement property can be obtained in all N modules by partially encoding the data in each of the N' modules and sending this encoded data to the remaining $N - N'$ modules. Notice that in contrast to the DJC algorithms, each of the N' modules calculates only one symbol of the error-correcting code and sends this symbol to all modules in the set of $N - N'$ modules.

Clearly in order to obtain agreement in the N' modules at least $3T+1$ modules need to be involved, but not all the modules involved need to execute

the decision-making process. Consequently the data received by the modules involved during round $K-2$, has only to be sent to the N' destinations represented by the set **D**. This reduces the information to be transmitted during round $K-1$. So in this case we are dealing with an *adapted* Interactive Consistency algorithm based on voting and coding, which is built from DJC algorithms and in which the set of destinations **D** is only a subset of the total set of modules **Ns**. The only difference between the "ordinary" IAC algorithm and the adapted IAC algorithm is in round $K-1$ and round K, i.e. during round $K-1$ messages are only sent to the modules in **D** and during round K the decision-making process is only executed in the destinations in **D**. The fact that the interactive consistency properties remain valid for the decisions calculated in the modules represented by **D** is obvious.

From the construction of the DJC algorithms we know that the amount of information which needs to be transmitted only depends on the error-correcting codes chosen. So reducing the set of destinations influences only the last round of information exchange, i.e. round $K-1$.

So let a system consist of N modules identified by the **Ns** of which at most T are faulty. This system first executes any adapted interactive consistency algorithm, which is chosen from the class of DJC algorithms

$$\mathcal{A}(T, K, 0, \mathbf{D}, \mathbf{Ns}) \quad \text{with:}$$

$$K = T+1 \quad \text{and} \quad |\mathbf{D}| \geq 2T+1 \quad \text{and} \quad |\mathbf{Ns}| \geq 3T+1 \tag{3.46}$$

This algorithm will lead to agreement among the correctly functioning modules of the set **D** about the original message in the source, $N' = |\mathbf{D}|$. In the N' modules a partial encoder function of a T-error-correcting code Z consisting of code words of n_z symbols and $n_z = |N'|$, is applied on the calculated decision. Thus, $Z(i)$ is applied on the decision in module i, with $i \in \mathbf{D}$. Notice that in this case the T-error-correcting code Z is determined by the N' sources and that in each of the sources one partial encoder function is applied. However in the Interactive Consistency algorithm based on voting and coding, the T-error-correcting coder is determined by the number of modules to which a modified message is to be sent, i.e the next-set $\mathbf{B}(\underline{s})$, and in one module n_c partial encoder functions are applied, one for each direction in which a message is to be sent.

In the Subset Method in the additional round of information exchange, round K, the partial encoded decisions are sent to all $N - N'$ remaining modules which each apply the decoder function $Z^{(-1)}$ on the received data. Because correct functioning modules in the set of **D** modules have reached agreement and there are no more than T faulty modules, the correct modules among the $N - N'$ modules arrive at the same decision as the N' modules did.

We will show that the Subset Method is often very efficient, though it has to be paid for by an additional round. In accordance with the terminology used by Dolev, we will call the subset of N' modules the "active modules" and the remaining modules the "passive modules". Notice however that in our Subset Method some of the passive modules may be involved in the broadcasting process of the interactive consistency algorithm.

Chapter 4

A comparison of the existing algorithms and the algorithms based on voting and coding

In this chapter the algorithms based on voting and coding will be compared with the existing synchronous deterministic algorithms. For this purpose a number of criteria will be defined and the resulting figures for a number of examples will be presented. We will show that for practical applications the algorithms in the class of algorithms which is based on voting and coding are favourable.

4.1 Introduction

In the previous chapter we have defined classes of Interactive Consistency algorithms based on voting and coding which fulfil the interactive consistency requirements.
The algorithms can be constructed for any set of parameters N, T, and K, which satisfy:

$$T \geq 1 \quad \text{and} \quad N \geq 3T + 1 \quad \text{and} \quad K = T + 1$$

So both the $N \geq 3T + 1$ and the $K \geq T + 1$ bound are met.

In this chapter we will compare these new classes of algorithms with the existing synchronous deterministic Interactive Consistency algorithms.

4.2 The algorithms selected for comparison

It cannot be said which of the algorithms is most favourable, an algorithm entirely based on voting, an algorithm based on coding or one of the other algorithms which are mentioned in the literature survey in Chapter 3, without knowing all design constraints.
In order to give some insight into the dependency of the parameters, a number of cases are presented in the Tables 4.1 to 4.5.

However, the selection of the algorithms which will be compared and the criteria on which they will be compared needs to be discussed first.

In our opinion a system of cooperating modules, in which the response time of a correctly functioning module might be unbounded, cannot be considered as a fault-tolerant system in the sense described in the first chapters of this thesis. Because the randomized algorithms may take an infinite time to come to a conclusion about what the source has sent, we will exclude them as algorithms which can be used in a (N, K)-fault-tolerant system. However, if we were to start from a more liberal definition of a fault-tolerant system, these algorithms could be very useful, but in that case, the algorithm itself would have to be considered as a source of failures. Further research in this area is needed.
Asynchronous deterministic algorithms do not exist, thus our comparison will be restricted to synchronous deterministic algorithms. This class again is divided into algorithms with and without authentication.
Among the synchronous deterministic algorithms based on authentication published so far, the one published by Dolev [Dolev 83-1], is favourable, because only $O(NT)$ messages have to be sent. Whether algorithms exist which require less messages is not known. Again this is a matter which awaits further investigations. Although authenticated algorithms are favourable in terms of the number of messages, their disadvantage is the authentication itself which might require great overheads in processing power, delay time and channel capacity.
From the preceding it follows that only the synchronous deterministic algorithms without authentication require further discussion.

The algorithm published in [Pease 80] and [Gils 85] is based on voting, and is contained in the class of algorithms described in Chapter 3. We will refer to this algorithm as the Pease algorithm.
The only remaining synchronous deterministic algorithm without authentication is the one published by Dolev [Dolev 82-3]. This algorithm requires a number of messages which is polynomial in N and T, while in the other algorithms the number of messages is exponential in N and T. We will refer to this algorithm as the Dolev algorithm.
So the algorithm by Pease, the algorithm by Dolev and the algorithms described in Chapter 3 should be included in the comparison. Among the latter special attention will be paid to the Minimal Voting algorithm and the Maximal Coding algorithm, these being two extremes in the design space.

4.3 The criteria

4.3.1 Introduction to the criteria

The criteria on which the algorithms will be compared are:

- The number of messages, *#mess*, that needs to be transmitted between the modules.
- The minimum size, *msize*, of the original message.

The amount of computation to be performed by the modules will not be taken into account because for all algorithms, in the same way, it is proportional to the number of transmitted messages. Moreover the computational effort will in general never be the bottleneck.

4.3.2 The number of messages in the algorithm based on voting and coding

For the class of algorithms which are based on voting and coding, the number of messages which has to be exchanged between the modules during the broadcast process, can be calculated as follows.
We will restrict ourselves to algorithms in which the choice of the code only depends on the round in which the encoding of the messages is performed.
Let a code used during round t consist of code words of $n_c(t)$ symbols, obtained from data words of $k_c(t)$ symbols, let the symbol size be $b_c(t)$

bits. Moreover let the amount of information of the original message to be broadcast by the source be the unit. So each amount of information is expressed in terms of a number of messages of the size of the original message $m(0)$.

During the first round the source broadcasts its data, after having it encoded, to $n_c(0)$ modules. Then during round 0, the amount of information which is sent in each direction is $(k_c(0))^{-1}$ unit messages, and the total amount of information which is transmitted in round 0 thus is $n_c(0).(k_c(0))^{-1}$. In round 1 the size of the message is again reduced by a factor $k_c(1)$ due to the partial encoder function. Hence the message size will be $(k_c(0).k_c(1))^{-1}$. Each message is sent in $n_c(1)$ directions, so the total amount of messages transmitted during round 1 will be $n_c(0).n_c(1)$. And thus the number of messages in terms of unit messages transmitted during round 1 is $n_c(0).n_c(1)(k_c(0).k_c(1))^{-1}$.

During the rounds $0 \leq t \leq K-2$, each module sends all its received data, after having it encoded, to $n_c(t)$ modules. Encoding for one direction, thus applying the partial encoder function, causes a data reduction of $k_c(t)$. The number of messages increases with a factor $n_c(t)$. Hence in round t,

$$\prod_{i=0}^{t} \left(\frac{n_c(i)}{k_c(i)}\right)$$

messages are transmitted.
The total amount of messages transmitted up to and including round $K-2$ then is

$$\sum_{t=0}^{K-2} \prod_{i=0}^{t} \left(\frac{n_c(i)}{k_c(i)}\right) \tag{4.1}$$

During the last round, i.e. round $K-1$, of the broadcast process, the messages are sent without encoding them, to all modules, which have not yet been passed by the message. During round $0 \cdots K-2$, each message has passed K distinct modules (the destination in round $K-2$ included). So each message resulting from round $(K-2)$, still has to be sent to $N-K$ modules. The total number of messages transmitted in round $(K-2)$, is $\prod_{i=0}^{K-2}(n_c(i))$. The size of the messages in round $K-1$ equals the size of the messages in round $K-2$ and is $\prod_{i=0}^{K-2}(k_c(i))^{-1}$. Thus the number of

messages which is transmitted in round $K-1$, is

$$(N-K).\prod_{t=0}^{K-2}\left(\frac{n_c(t)}{k_c(t)}\right) \tag{4.2}$$

From the preceding follows that:
For the algorithm which is based on voting and coding, the total amount of messages which is transmitted during the broadcast process, in terms of the number of unit messages, is:

$$\#mess = (N-K).\prod_{t=0}^{K-2}\left(\frac{n_c(t)}{k_c(t)}\right) + \sum_{t=0}^{K-2}\prod_{i=0}^{t}\left(\frac{n_c(i)}{k_c(i)}\right) \tag{4.3}$$

Notice that because $K = T+1$, the number of messages is exponential in T. From equation (4.3) it is readily seen that if N and T are fixed, the number of messages is minimal for the smallest possible fractions $(\frac{n_c(t)}{k_c(t)})$.

4.3.3 The number of messages in the Subset Method

In the description of the Subset Method we maintained the relation $K = T+1$, hence due to the additional round, the number of rounds in the Subset Method is $K+1$ and the number of rounds of information exchange is K.

So let a system consist of N modules identified by the set **Ns** of which at most T are faulty. This system first executes during the rounds $0, \cdots K$ any adapted interactive consistency algorithm, which is chosen from the class of DJC algorithms

$$\mathcal{A}(T, K, a, \mathbf{D}, \mathbf{Ns}) \quad \text{with:}$$

$$K = T+1 \quad \text{and} \quad |\mathbf{D}| \geq 2T+1 \quad \text{and} \quad |\mathbf{Ns}| \geq 3T+1 \tag{4.4}$$

The destinations of this algorithm which are given by the set **D** are called the active modules. Let $N = |\mathbf{Ns}|$ and $N' = |\mathbf{D}|$.
After the decisions have been calculated in the active modules, the results are encoded and are sent during round K to the remaining $N-N'$ modules.

The number of messages transmitted during the round $0 \cdots K-2$, for the algorithm based on voting and coding turned out to be

$$\sum_{t=0}^{K-2} \prod_{i=0}^{t} \left(\frac{n_c(i)}{k_c(i)}\right) \tag{4.5}$$

(cf (4.1))
This figure is independent of the number of modules N, be it that the values $n_c(t)$ are restricted by N.

Not all modules in the system which are represented by the set **Ns** need to be really involved in the Interactive Consistency algorithm. There might well be modules which never receive a message during the rounds $0, \cdots K-1$. Let the modules which are involved in the algorithm during the rounds $0, \cdots, K-2$ be denoted by the set $\mathbf{Ns}_{inv}$ and let $N_{inv} = |\mathbf{Ns}_{inv}|$. Notice that the rounds $K-1$ and K are excluded from the definition of the modules involved. Clearly $N_{inv} \leq N$.
The number N_{inv} of modules involved in the rounds $0, \cdots, K-2$ is calculated as follows:
The number of modules involved is again kept as small as possible during the successive broadcast rounds. Hence at the end of round 0 the number of modules involved is $n_c(0)+1$. At the end of round 0 each message has passed 2 modules, the destination in round 0 included, so the number of modules involved needed at the end of round 1 is $\max\big((n_c(0)+1), (n_c(1)+2)\big)$. At the end of round $t-1$ each message has passed $t+1$ modules, the destination in round $t-1$ included, so the number of modules involved needed at the end of round t must be at least $(n_c(t)+t+1)$. And thus

$$N_{inv} = \mathrm{Max}\big(0 \leq t \leq K-2 \ : \ n_c(t)+t+1\big) \tag{4.6}$$

During the last round, i.e. round $K-1$, of the broadcast process of the Interactive Consistency algorithm, the messages are sent without encoding them, to all N' active modules the set **D**, which are not yet passed by the message. It is always possible to construct the algorithm such that either these N' active modules are a subset of the N_{inv} modules involved in the interactive consistency algorithm, or the N_{inv} modules are a subset of the N' active modules. Thus we assume

$$\mathbf{D} \subset \mathbf{Ns}_{inv} \quad \text{or} \quad \mathbf{Ns}_{inv} \subset \mathbf{D}$$

Moreover we assume that the source is contained in the set of active modules and in the set of modules involved.

Suppose $\mathbf{D} \subset \mathbf{Ns}_{inv}$.
Thus $N' \leq N_{inv}$.
During round $0 \cdots K-2$, each message has passed K distinct modules (the source module and the destination in round $K-2$ included). Obviously these K modules belong to the set $\mathbf{Ns}_{inv}$. However from these K modules $K-1$ might belong to the set $\mathbf{Ns}_{inv} - \mathbf{D}$, because we assumed $a \in \mathbf{D}$. So each message received in round $K-2$ is sent during round $K-1$ to at most $N'-1$ other modules. If we ingnore the fact that it is not necessary to send messages to involved but not active modules, then the number of modules to which a message is to be sent is at most $N_{inv} - K$. Hence the number of modules to which a message has to be sent during round $K-1$ will never be more than $\min\big((N'-1),(N_{inv}-K)\big)$.

Next suppose $\mathbf{Ns}_{inv} \subset \mathbf{D}$.
Thus $N_{inv} \leq N'$.
At the end of round $K-2$, each message has passed K modules of the set $\mathbf{D}$ of N' modules, hence during round $K-1$ each message needs to be sent to $N'-K$ modules.

So in conclusion, the number of modules to which a message has to be sent during round $K-1$ is not more than:

$$\text{if } N' \leq N_{inv} \text{ then } \min\big((N'-1),(N_{inv}-K)\big)$$

and

$$\text{if } N' \geq N_{inv} \text{ then } N'-K$$

Notice that $N'-K \geq \min\big((N'-1),(N_{inv}-K)\big)$ implies that either $(N'-K) \geq (N'-1)$ or $(N'-K) \geq (N_{inv}-K)$. Because $K \geq 2$ the first term never holds and thus $N'-K \geq \min\big((N'-1),(N_{inv}-K)\big)$ implies $N' \geq N_{inv}$.
Similarly, $N'-K \leq \min\big((N'-1),(N_{inv}-K)\big)$ implies that $(N'-K) \leq (N'-1)$ and $(N'-K) \leq (N_{inv}-K)$. Because $K \geq 2$ the first term always holds and thus $N'-K \leq \min\big((N'-1),(N_{inv}-K)\big)$ implies $N' \leq N_{inv}$.

Hence the number of modules to which a message has to be sent during round $K-1$ is not more than

$$\max\big((N'-K),\min\big((N'-1),(N_{inv}-K)\big)\big)$$

The total number of messages transmitted in round $(K-2)$, is $\prod_{i=0}^{K-2}(n_c(i))$. The size of the messages in round $K-1$ equals the size of the messages in round $K-2$ and is $\prod_{i=0}^{K-2}(k_c(i))^{-1}$. Thus the number of messages which is transmitted in round $K-1$, is at most

$$\Big(\max\big((N'-K),\min\big((N'-1),(N_{inv}-K)\big)\big)\Big).\prod_{t=0}^{K-2}\Big(\frac{n_c(t)}{k_c(t)}\Big) \tag{4.7}$$

The additional number of messages for the extra round in the Subset Method is calculated as follows:
Agreement after $K=T+1$ rounds is obtained in N' modules. At the end of round $T+1$, thus after the decision-making process, each module calculates its symbol of the T-error-correcting code, with parameters $n_c = N'$ and $k_c = n_c - 2T$. Hence we restricted ourselves for simplicity to the class of MDS codes. The result of the decision-making process is of the same type as the original message in the source, hence the size of the symbols resulting from the encoding is $1/(N'-2T)$ unit message.
In the additional round, round $T+2$, the N' modules send their symbol to the $N-N'$ passive modules. Thus the number of additional unit messages is:

$$\frac{N'(N-N')}{N'-2T} \tag{4.8}$$

From the preceding equations (4.5), (4.7), (4.6), and (4.8) it follows that:
In the Subset Method the total amount of information which is transmitted during the broadcast process, in terms of the number of unit messages, is at most:

$$\begin{aligned}\#mess \;=\; & \frac{N'(N-N')}{N'-2T} \;+\; \sum_{t=0}^{K-2}\prod_{i=0}^{t}\Big(\frac{n_c(i)}{k_c(i)}\Big) \\ & +\Big(max\big((N'-K),min\big((N'-1),(N_{inv}-K)\big)\big)\Big).\prod_{t=0}^{K-2}\Big(\frac{n_c(t)}{k_c(t)}\Big)\end{aligned} \tag{4.9}$$

with

$$N_{inv} = Max\big(0 \le t \le K-2 \;:\; n_c(t)+t+1\big) \tag{4.10}$$

and

$$K = T+1$$

4.3.4 The minimum size of the original message in the source

When discussing the Interactive Consistency algorithms on voting and coding, we already pointed out that in a module q, the size of the message on which a partial encoder function operates, thus the size of the data word, is $k_{(\underline{u},p,q)}.b_{(\underline{u},p,q)}$ bits, in which $k_{(\underline{u},p,q)}$ is the number of data symbols of the code $\mathcal{Y}_{(\underline{u},p,q)}$, and $b_{(\underline{u},p,q)}$ is the size of the symbols expressed in the number of bits.
The size of the data word received by module q must be equal to the message (symbol) size $b_{(\underline{u},p)}$, produced by the partial encoder functions of the code $\mathcal{Y}_{(\underline{u},p)}$ in module p. And thus

$$b_{(\underline{u},p)} = k_{(\underline{u},p,q)}.b_{(\underline{u},p,q)}$$

We also concluded that the choice of the code only needs to depend on the round t. Moreover $|(\underline{u},p)| = t+1$. So instead of parametrizing the code with the string $(\underline{u},p)$ which identifies the message on which the DJC algorithms $A_{K-t}((\underline{u},p),d)$ operate, we may parametrize the code with the round t. So $\mathcal{Y}_t$ denotes the code applied by the algorithms $A_{K-t}((\underline{u},p),d)$. The number of symbols in a code word then is denoted by $n_c(t)$, The number of symbols in a data word by $k_c(t)$, and the symbol size by $b_c(t)$.
And thus we require

$$\forall t \ : \ 1 \leq t \leq K-2 \Longrightarrow b_c(t-1) = k_c(t).b_c(t) \tag{4.11}$$

and for the original message in the source

$$msize = k_c(0).b_c(0) \tag{4.12}$$

Moreover the symbol size of each code must be sufficiently large, so in the case of MDS codes we need to satisfy

$$\begin{aligned} \forall t \ : \ 0 \leq t \leq K-2 \Longrightarrow \big(& (k_c = 1 \Longrightarrow b_c(t) \geq 1) \wedge \\ & (k_c \geq 2 \Longrightarrow b_c(t) \geq \log_2(n_c(t)-1)) \big) \end{aligned} \tag{4.13}$$

By means of these relations the value *msize* can be calculated.

The Subset Method might impose an additional constraint. In the Subset Method in the additional round, round $T+1$, the messages which result

from the decision-making process in the N' modules are encoded with a $(N', N' - 2T, 2T + 1)$-MDS code. Hence the size of the original message must equal the size of the data word of this code. The symbol size of this code should be at least $\lceil \log_2(N' - 1) \rceil$ bits. Consequently the size of the data word is at least $(N' - 2T).\lceil \log_2(N' - 1) \rceil$ bits. So the Subset Method additionally requires

$$N' = 2T + 1 \Longrightarrow msize \geq 1$$
$$N' \geq 2T + 1 \Longrightarrow msize \geq (N' - 2T).\log_2(N' - 1) \tag{4.14}$$

4.3.5 The number of messages in the Dolev-algorithm

In order to minimize the number of messages needed by the algorithm, in [Dolev 82-3], Dolev divides the system into two sets of modules, the active modules and the passive modules. The active modules include the source and the number of active modules is $3T+1$. Thus the active modules form a minimal system with respect to T. The IAC algorithm in fact only runs on the active modules, while the passive modules only receive that data which allows them to obtain agreement.

We will first concentrate on the active modules, thus we assume $N = 3T+1$.

In the first round a binary value is sent to all modules. This will result in $N - 1$ messages of one bit. During the following rounds, the data which can be broadcast by each module, consists of a number of messages. Each of these messages may be the name of a module, a "supporter", or a special message called " * ", which means that the sender "initiated" in the previous round. So a message can be encoded by $\lceil \log_2(N + 1) \rceil$ bits. A module sends each module name at most once during the entire process. Also the " * "-message is sent at most once by a module. Thus the total number of messages sent by a particular module is at most $N + 1$. These messages are sent to all other modules. Thus the total number of one-bit messages needed by the algorithm, including round 0, is at most:

$$N - 1 + (N - 1)N(N + 1)\lceil \log_2(N + 1) \rceil$$

Or, because $N = 3T + 1$:

$$3T + 3T(3T + 1)(3T + 2)\lceil \log_2(3T + 2) \rceil$$

This figure differs slightly from the figure calculated in [Dolev 82-3], because Dolev also counts the messages sent by a module to itself and ignores the messages sent in round 0.

If $N > 3T + 1$, the set of passive modules is non-empty. These passive modules only receive the " * " messages from the active modules and the passive modules do not have to intercommunicate. Each module broadcasts the " * " message only once during the algorithm. So the number of additional one-bit messages is only $(3T + 1)(N - 3T - 1)$.
From the preceding it follows that:
The total number of messages needed by the Dolev-algorithm is at most:

$$(3T + 1)(N - 3T - 1) + 3T + 3T(3T + 1)(3T + 2)\lceil \log_2(3T + 2) \rceil$$

Or:

$$\#mess = (3T + 1)(N - 3T) - 1 + 3T(3T + 1)(3T + 2)\lceil \log_2(3T + 2) \rceil \quad (4.15)$$

4.4 The algorithms compared

The criteria "number of messages", the number of rounds K, and the minimal size of the original message in the source are calculated for different values N and T, for

- the selected existing algorithms, i.e.:
  - the Pease algorithm and
  - the Dolev algorithm.
- the algorithms developped and proved in this thesis, i.e.:
  - the Minimal Voting algorithm,
  - the Maximal Coding algorithm, and

- some arbitrarily chosen algorithms based on voting and coding models.

Moreover, some examples of the Subset Method are presented in which the IAC algorithm runs on a smaller number of active processors and the remaining modules receive their data encoded in an additional round. Dividing the modules in active and passive modules in order to reduce the number of messages is a method already implemented in the Dolev algorithm.
The results are presented in the tables 4.1, 4.2, 4.3, and 4.4. Table 4.5 compares separately the Dolev algorithm, the Minimal Voting algorithm and the Maximal Coding algorithm.

Notice that in these tables the number of messages are exact figures for the algorithm based on voting. The number of messages calculated for the Subset method and the Dolev algorithm is an upperbound and in general will be a little less. The calculated minimum message size is in all cases the exact value, but by introducing dummy bits in the data word often the message size can be reduced on the account of an increased number of unit messages.

In the tables, the algorithms based on voting and coding are identified by the error-correcting codes which are applied consecutively during the broadcast process and if applicable by the code used during the additional round in the Subset Method. Each code is indicated by a $[n_c, k_c, b_c]$-tuple in which n_c stands for the number of symbols in the code word, k_c stands for the number of symbols in the corresponding data word, and b_c stands for the number of bits of which each symbol is composed. All codes mentioned have a Hamming distance of least $2T + 1$ and do exist. In most cases MDS codes are applied, but in some cases we used other codes in our examples, like Hamming codes and BCH codes. In cases where non-MDS codes are applied, the design constraints mentioned in the previous sections have been adapted accordingly.
During round $K - 1$ no code is applied but instead all data is sent to all modules which have not yet been passed. Therefore where the Subset Method is applied this round is indicated by "-".

The examples presented in the tables are not exhaustive and are only intended to explore the design space. Many other and probably even better solutions can be found. The purpose of the data presented is only to give an

indication of the way in which the parameters and the criteria are related.

Comparing algorithms with a different minimal message size is rather dangerous, because

- If the minimum message size becomes large the overhead caused by authentication becomes relatively small, and thus the algorithms presented in the following tables can no longer compete with synchronous deterministic algorithms based on authentication.
- It has not been investigated whether it is possible to generalize the Dolev algorithm to larger message sizes. Clearly if the message size is l we are able to apply the algorithm l times. This does not influence our comparison. However a non-trivial generalization of the Dolev algorithm might result in more efficient algorithms in terms of the number of unit messages.

From the tables the following conclusions may be drawn:

- If the number of faults which needs to be tolerated in the system is four or more, $T \geq 4$, and the number of modules is minimal, i.e. $N = 3T + 1$, all algorithms are extremely inefficient. This means each message which has to be distributed by a single source to all other modules generates many thousands of new messages that must be transmitted over the communication network between the modules. In practical applications, synchronous deterministic algorithms for $T \geq 4$ therefore cannot be implemented.
- If $T = 4$ the Maximal Coding algorithm still is a factor 2 better than the Dolev algorithm, however on account of the minimal message size. All other algorithms are at least a factor 10 worse. For $T \geq 5$ the Dolev algorithm is the best known synchronous deterministic algorithm, but unfortunately it is only of theoretical interest.
- If the number of modules needs to be minimal with respect to the number of tolerable faults, i.e. $N = 3T + 1$, and the minimal size of the original message must be one, in practical systems, i.e. $T \leq 3$, the Minimal Voting algorithm requires the least number of messages.
- In practical systems, i.e. $T \leq 3$, the Maximal Coding Algorithm turns out to be superior in terms of number of messages. However this is

on account of the minimal message size, which is some cases might become unacceptably large.

- If $N > 3T + 1$ and $T \leq 3$, and an acceptable minimal message size is required, other codes can be chosen such that the resulting algorithm has an acceptable message size and a number of messages which is not to far away from the optimum.
- Especially if $N >> 3T + 1$, $T = 2$ or 3, and the minimal message size should be not too large, further improvement can be obtained by applying the Subset Method, in which the modules are divided into in active and passive modules and the Interactive Consistency algorithm is executed on the active modules, while the passive modules receive the encoded decision of the active modules in an additional round.
- If $T = 1$ and $N = 4$ the Pease algorithm, the Minimal Voting and the Maximal Coding algorithm turn out to be the same.

The previous discussion shows that many items need further research. The $K \geq T+1$ bound and the $N \geq 3T+1$ bound restrict the class of Interactive Consistency algorithms, but algorithms which require fewer messages to be transmitted compared to the ones presented here, might very well exist. Notice that a bound for the minimum number of messages has not yet been found.

For the following tables holds:

$N_{tot.} =$ *the total number of modules in the system.*

$N_{act.} =$ *the number of modules in the system in which in a case where the subset method is applied, the decision-making process of the IAC algorithm is executed during the last but one round.*

$K =$ *the number of rounds of information exchange.*

$msize =$ *the minimal size of the original message in the source in bits.*

$\#mess =$ *the number of messages transmitted by the algorithms counted according to the size of the original message in the source.*

$T = 1$						
algorithm	$N_{tot.}$	N_{act}	K	*msize*	*#mess*	applied codes
Dolev	4	4	5	1	183	
Pease	4	4	2	1	9	[3,1,1]
MinVot	4	4	2	1	9	[3,1,1]
MaxCod	4	4	2	1	9	[3,1,1]
Dolev	5	4	5	1	187	
Pease	5	5	2	1	16	[4,1,1]
MinVot	5	5	2	1	12	[3,1,1]
MaxCod	5	5	2	4	8	[4,2,2]
Subset	5	4	3	4	11	[3,1,4]-[4,2,2]
Dolev	6	4	5	1	191	
Pease	6	6	2	1	25	[5,1,1]
MinVot	6	6	2	1	15	[3,1,1]
	6	6	2	4	10	[4,2,2]
MaxCod	6	6	2	6	8.3	[5,3,2]
Subset	6	5	3	6	9.7	[4,2,3]-[5,3,2]
Subset	6	4	3	4	13	[3,1,4]-[4,2,2]
Dolev	16	4	5	1	231	
Pease	16	16	2	1	225	[15,1,1]
MinVot	16	16	2	1	45	[3,1,1]
	16	16	2	4	30	[4,2,2]
	16	16	2	18	20	[8,6,3]
	16	16	2	4	26.25	[7,4,1]
	16	16	2	11	20.5	[15,11,1]
MaxCod	16	16	2	52	17.3	[15,13,4]
Subset	16	3	3	1	48	[3,1,1]-[3,1,1]
Subset	16	3	3	4	45	[4,2,2]-[3,1,4]
Subset	16	4	3	4	33	[3,1,4]-[4,2,2]
Subset	16	4	3	4	32	[4,2,2]-[4,2,2]
Subset	16	8	3	30	20.4	[7,5,6]-[8,6,5]
Pease	64	64	2	1	3969	[63,1,1]
Dolev	64	4	5	1	423	
MinVot	64	64	2	1	189	[3,1,1]
	64	64	2	4	126	[4,2,2]
	64	64	2	32	78.8	[10,8,4]
	64	64	2	4	110.25	[7,4,1]
	64	64	2	57	69.6	[63,57,1]
MaxCod	64	64	2	336	65.1	[63,61,6]
Subset	64	4	3	4	129	[3,1,4]-[4,2,2]
Subset	64	3	3	1	192	[3,1,1]-[3,1,1]
Subset	64	18	3	80	72	[18,16,5]-[18,16,5]

Table 4.1: *The minimal message size and number of messages generated by some IAC algorithms which tolerate one failing module.*

$T = 2$						
algorithm	$N_{tot.}$	N_{act}	K	*msize*	*#mess*	applied codes
Dolev	7	7	7	1	1014	
Pease	7	7	3	1	156	[6,1,1][5,1,1]
MinVot	7	7	3	1	130	[5,1,1][5,1,1]
MaxCod	7	7	3	6	78	[6,2,3][5,1,3]
Dolev	8	7	7	1	1021	
Pease	8	8	3	1	259	[7,1,1][6,1,1]
MinVot	8	8	3	1	155	[5,1,1][5,1,1]
	8	8	3	12	57	[6,2,6][6,2,3]
MaxCod	8	8	3	18	44.3	[7,3,6][6,2,3]
Subset	8	5	4	1	145	[5,1,1][5,1,1]-[5,1,1]
Pease	16	16	3	1	2955	[15,1,1][14,1,1]
Dolev	16	7	7	1	1077	
MinVot	16	16	3	1	355	[5,1,1][5,1,1]
	16	16	3	66	45.9	[15,11,6][14,6,1]
MaxCod	16	16	3	440	28.1	[15,11,40][14,10,4]
Subset	16	5	4	1	185	[5,1,1][5,1,1]-[5,1,1]
Subset	16	8	4	24	60.3	[7,3,8][6,2,4]-[8,6,4]
Pease	64	64	3	1	242235	[63,1,1][62,1,1]
Dolev	64	7	7	1	1413	
MinVot	64	64	3	1	1555	[5,1,1][5,1,1]
	64	64	3	2950	83	[63,59,50][62,50,1]
MaxCod	64	64	3	20532	72	[63,59,348][62,58,6]
Subset	64	5	4	1	425	[5,1,1][5,1,1]-[5,1,1]
Subset	64	12	4	48	120	[8,4,12][8,4,3]-[12,8,6]
Subset	64	16	4	660	92	[15,11,60][14,10,6]-[16,12,55]

Table 4.2: *The minimal message size and number of messages generated by some IAC algorithms which tolerate two failing modules.*

$T = 3$						
algorithm	$N_{tot.}$	N_{act}	K	*msize*	*#mess*	applied codes
Dolev	10	10	9	1	3969	
Pease	10	10	4	1	3609	[9,1,1][8,1,1][7,1,1]
MinVot	10	10	4	1	2457	[7,1,1][7,1,1][7,1,1]
MaxCod	10	10	4	18	603	[9,3,6][8,2,3][7,1,3]
MaxCod	11	11	4	72	250	[10,4,18][9,3,6][8,2,3]
Pease	16	16	4	1	35715	[15,1,1][14,1,1][13,1,1]
Dolev	16	10	9	1	4029	
MinVot	16	16	4	1	4515	[7,1,1][7,1,1][7,1,1]
	16	16	4	256	212	[10,4,64][10,4,16][10,4,4]
MaxCod	16	16	4	2016	75	[15,9,224][14,8,28][13,7,4]
Subset	16	10	5	16	2472	[7,1,16][7,1,16][7,1,16]-[10,4,4]
Subset	16	10	5	256	180	[10,4,64][10,4,16][10,4,4]-[10,4,64]
Dolev	64	10	9	1	4509	
MinVot	64	64	4	1	20979	[7,1,1][7,1,1][7,1,1]
MaxCod	64	64	4	1053360	85	[63,57,18480][62,56,330][61,55,6]
Subset	64	10	5	256	300	[10,4,64][10,4,16][10,4,4]-[10,4,64]

Table 4.3: *The minimal message size and number of messages generated by some IAC algorithms which tolerate three failing modules.*

$T = 4$						
algorithm	$N_{tot.}$	N_{act}	K	*msize*	*#mess*	applied codes
Pease	13	13	5	1	108384	[12,1,1][11,1,1][10,1,1][9,1,1]
Dolev	13	13	11	1	8748	
MinVot	13	13	5	1	59868	[9,1,1][9,1,1][9,1,1][9,1,1]
MaxCod	13	13	5	96	4524	[12,4,24][11,3,8][10,2,4][9,1,4]
Dolev	16	13	11	1	8787	
	16	16	5	1024	1011	[12,4,256][12,4,64][12,4,16][12,4,4]
MaxCod	16	16	5	3360	488	[15,7,480][14,6,80][13,5,16][12,4,4]
Dolev	64	13	11	1	9411	
	64	64	5	1024	4899	[12,4,256][12,4,64][12,4,16][12,4,4]
Subset	64	16	6	1024	1107	[12,4,256][12,4,64][12,4,16] [12,4,4]-[16,8,128]

Table 4.4: *The minimal message size and number of messages generated by some IAC algorithms which tolerate four failing modules.*

$N = 3T + 1$					
		Dolev $msize = 1$	MinVot $msize = 1$	MaxCod	
N	T	$\#mess$	$\#mess$	$msize$	$\#mess$
4	1	183	9	1	9
7	2	1014	130	6	78
10	3	3969	2457	18	603
13	4	8748	59868	96	4524
16	5	20415	$1.8\ 10^6$	480	33356

Table 4.5: *A comparison of the minimal message size and the number of messages generated by the Dolev (polynomial) algorithm, the Minimal Voting (exponential) algorithm, and the Maximal Coding (exponential) algorithm, in the case where* $N = 3T + 1$.

Chapter 5

Interconnecting fault-tolerant systems

In this chapter the solution to the Input Problem will be presented. The correctness of the behaviour of a fault-tolerant system depends among other things on the correct distribution of the data descending from unreliable I/O devices over the modules of the fault-tolerant system. A maliciously behaving system, whether it is fault-tolerant or not, should never defeat a correctly functioning fault-tolerant system, i.e a system which does not contain more faulty modules than it is designed to tolerate. In order to cope with this problem, in this chapter the definition of interactive consistency will be reformulated for interactive consistency between communicating fault-tolerant systems and a number of interconnection methods and algorithms will be presented which solve this problem. These interconnection methods and algorithms are based on the Dispersed Joined Communication algorithms and the Interactive Consistency algorithms. The implementation of such an algorithm in a (4,2) concept fault-tolerant computer system is described in detail.

5.1 Introduction

In the introduction, Chapter 1, Section 1.5, we already pointed out that fault-tolerant systems always will be connected to other systems based on different methods for reliability improvement. In any case they will be connected to basically unreliable input devices. For example, the error registers

which store the information about the faults detected by the decoders in a (N, K) concept fault-tolerant system are to be considered as such unreliable input devices.
The interconnection of these sources to a fault-tolerant system has to be done very carefully.
This means two communicating systems must never defeat each other as long as they are both functioning correctly. Recall from Chapter 2 that a fault-tolerant system is functioning correctly if its external behaviour in the presence of a number of tolerable internal faults is equivalent to the behaviour expressed by the specification. So data originating from a malfunctioning system must never cause the receiving system to go down. In Section 1.5 we showed that an external system generating broadcast faults might cause the receiving fault-tolerant system to go down, even if the number of faults in the receiving system is not more than it is designed to tolerate.
In Chapter 2 the discussion of the fault-tolerance properties of the N-modular redundant systems and the systems based on generalized masking, i.e. the (X, Y, T) and the (X, Y, Z, T) fault-tolerant systems assumed that the distribution function X is functioning correctly. Due to broadcast faults, in a real system this requirement will however not be satisfied by just transmitting the output value of a module to a number of destinations. We will have to cope in some way with these broadcast faults, which have been defined in Chapter 1, and thus the distribution function will have to be implemented by some algorithm.

In this chapter we will present four methods for the interconnection of fault-tolerant systems with each other and with single unreliable I/O devices. These methods are based on the Interactive Consistency algorithms or the Dispersed Joined Communication algorithms which have been described in Chapter 3. We will show that an (N, K) concept fault-tolerant computer which is interconnected with its environment according to one of these methods never can be brought down by a maliciously behaving system transmitting data to it, even if the transmitting system is a maliciously behaving fault-tolerant system, [Krol 85].

The following methods and system environments will discussed

- The DJC Method applied to a single input device.
- The DJC Method applied to a fault-tolerant input device with post-

observation.

- The DJC Method applied to a fault-tolerant input device with pre-observation.

- The DJC Method applied to a NMR fault-tolerant input device with pre-observation and pre-coding.

The first three methods are similar to the methods published in [Krol 85] and [Krol 86] in which these methods have been explained on the basis of Interactive Consistency algorithms.

5.2 Communication of a fault-tolerant system with its environment

In this section we will deal with the problem of the communication of a fault-tolerant system with its environment. It is reasonable to require that a correctly implemented fault-tolerant system only goes down if more modules in it behave maliciously than the system is designed to tolerate. Recall that in Section 2.5 we concluded that a $(\mathcal{X}, \mathcal{Y}, \mathcal{Z}, T)$ fault-tolerant system is functioning correctly if it is correctly initialized and not more than T of its modules behaved maliciously in any period between two complete re-initializations. Similarly an (N, K) concept fault-tolerant computer which is able to tolerate T maliciously behaving modules and which is correctly initialized must continue behaving correctly as long as the number of maliciously behaving modules is less or equal than T, regardless of the behaviour of the environment. Recall from Section 2.3.7 that the meaning of "correct behaviour" should not be interchanged with the "correctness" of data values. In Chapter 2 we showed that correct behaviour which is independent of the environment indeed can be obtained if the distribution function $\mathcal{X}$ is functioning correctly.
Moreover recall that the correctness of the decoding function $\mathcal{Y}^{(-1)}$ is the reponsibility of the observer.
In this section we will present four generally applicable algorithms which are executed by the transmitting and the receiving fault-tolerant system and which result in the correct execution of the function $\mathcal{X}$, provided sufficient modules of the receiving system are functioning correctly.

We assume that our universe of discourse consists of a receiving system, which is called the r-system and its environment which is called the (transmitting) t-system. Let the receiving fault-tolerant r-system consist of N_r modules and let it be able to tolerate the influence of T_r maliciously behaving modules. This system receives data from a possibly "fault-tolerant" t-system, which consists of N_t modules and is able to tolerate the influence of T_t faulty modules. The t-system may be any system, so for instance $N_t = 1$ and $T_t = 0$. Thus a t-system which is composed of a single module is also allowed. The observing function of the t-system is denoted by $yt^{(-1)}$. Obviously if $T_t = 0$ then $yt^{(-1)}$ is an identity function.

We define *interactive consistency between communicating fault-tolerant systems* as follows:

Definition 5.1

- *Let a receiving system, called the r-system consist of N_r modules. The r-system is designed to tolerate T_r maliciously behaving modules and let it contain at most T_r maliciously behaving modules.*
- *Let a transmitting system, called the t-system consist of N_t modules. The t-system is designed to tolerate T_t maliciously behaving modules.*
- *Let the t-system transmit an (encoded) message y to the r-system. Let the original data value be x_t. The N_t-tuple y thus is the encoded version of the of the message x_t. The partial encoder functions are denoted by $yt(i_t)$ with $i_t \in \mathbf{Ns}_t$, where $\mathbf{Ns}_t$ is the set of modules constituting the t-system. Each correctly functioning module i_t of the t-system sends a partially encoded message $y(i_t)$ to all or some of the modules of the r-system, in which $y(i_t) = yt(i_t)(x_t)$.*

Then interactive consistency between communicating systems is guaranteed if:

- *The well-functioning modules d_r of the r-system always agree with each other on the decoded message $dec(d_r)$ they calculate from the data they received from the t-system.*
- *If the t-system is functioning correctly, i.e. if it contains at most T_t maliciously behaving modules, the above-mentioned agreement should equal the decoded data actually sent, i.e. $dec(d_r) = x_t$.*

Thus, if $\overline{\mathbf{F}}_t$ *and* $\overline{\mathbf{F}}_r$ *represent the sets of correctly functioning modules in the t-system and the r-system respectively, then:*

$$\forall \overline{\mathbf{F}}_t, \overline{\mathbf{F}}_r \ : \ |\overline{\mathbf{F}}_r| \geq N_r - T_r \Longrightarrow$$
$$\Big(\big(|\overline{\mathbf{F}}_t| \geq N_t - T_t \Longrightarrow [\ \forall d_r \ : \ d_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = x_t \ ] \big) \wedge$$
$$[\ \forall d_r, e_r \ : \ d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = dec(e_r) \ ] \Big)$$

□

Without proof we state that these requirements can only be satisfied when the number of modules in the receiving system is larger than three times the number of malfunctioning modules in that system, thus only if $N_r \geq 3T_r + 1$.

5.2.1 The DJC Method applied to a single input device

IF the t-system consists of one module then the DJC Method applied to a single input device is defined as follows:

Definition 5.2

- *The single module* 0_t *in the t-system communicates its message* $m(0_t) = x_t$ *to all modules* d_r *of the r-system by means of a DJC algorithm from the class* $\mathcal{A}(T_r, T_r + 2, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$.
- *The decision* $dec(d_r)$ *in a module* d_r *of the r-system is the result* $dec_{T_r+2}((0_t), d_r)$ *of the DJC algorithm.*

□

The method is elucidated in Figure 5.1. In this figure the DJC algorithm from the class $\mathcal{A}(T_r, T_r + 2, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$ has been decomposed in algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ with $a_r \in \mathbf{B}(0_t)$, surrounded by the encoding function $y_{(0_t)}$ and the decoder function $y^{(-1)}_{(0_t)}$. The communications due to the algorithms from the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ are denoted by o o o.

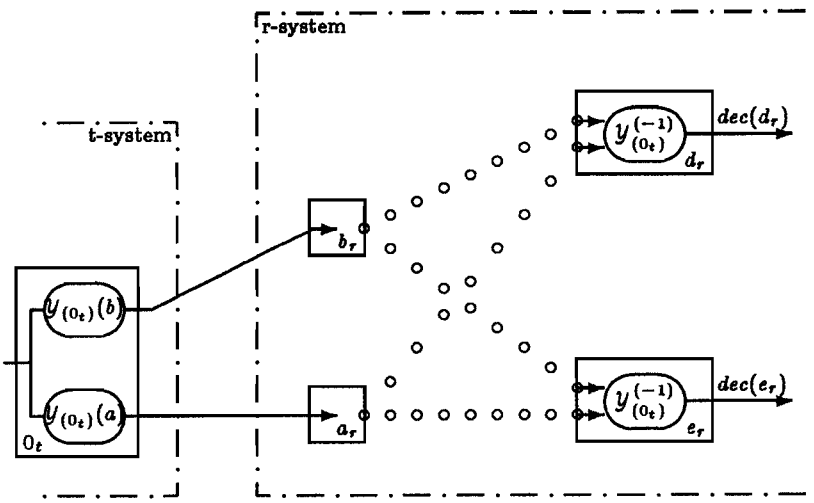


Figure 5.1: *The DJC Method applied to a single input device*

This decomposition corresponds to the decomposition which has been used in the recursive definition of the DJC algorithms in Chapter 3.

Notice that the algorithms from the classes $\mathcal{A}(T_r, T_r+1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ with $a_r \in \mathbf{B}(0_t)$ are IAC algorithms according to the definition in Section 3.4. Futhermore from the definition of the class $\mathcal{A}(T_r, T_r+2, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$ we know that $|\mathbf{B}(0_t)|$ should be at least $2T_r+1$. Hence our algorithm can be implemented if $N_r \geq 3T_r+1$.

A single input device which is connected to a fault-tolerant system according to the method defined above can never cause this system to go down, i.e.:

Theorem 5.1 *If the DJC Method is applied for connecting a single input device, the t-system, to a fault-tolerant system, the r-system, as is defined in Definition 5.2, then the t-system and the r-system satisfy the interactive consistency requirement between communicating systems as is defined in Definition 5.1.* □

Proof:

From Definition 5.2 we know that the original message $m(0_t)$ in the input device is communicated to the modules d_r of the r-system by means

of DJC algorithms from the class $\mathcal{A}(T_r, T_r + 2, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$. From Theorem 3.2 on page 143 we know that if the modules 0_t and d_r are both functioning correctly, the result $dec_{T_r+2}((0_t), d_r)$ equals the original message $m(0_t)$ and thus equals x_t. Thus:

$$dec_{T_r+2}((0_t), d_r) = x_t$$

Since $T_t = 0$ it holds that the proposition "0_t is functioning correctly" is equivalent to the condition $|\overline{\mathbf{F}}_t| \geq N_t - T_t$. Moreover we defined $dec(d_r) = dec_{T_r+2}((0_t), d_r)$ and thus

$$|\overline{\mathbf{F}}_t| \geq N_t - T_t \Longrightarrow [\ \forall d_r\ :\ d_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = x_t\ ] \qquad (5.1)$$

Which proves the first part of the "interactive consistency between communicating systems".

From the second part of Theorem 3.2 we know that if a message is communicated by means of an algorithm from the class $\mathcal{A}(T_r, T_r+2, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$ to the modules in $\mathbf{Ns}_r$, the following holds:
If the modules d_r and e_r, with $d_r, e_r \in \mathbf{Ns}_r$, are both functioning correctly and the results of the algorithm calculated in module d_r and e_r are unequal then the number of faulty modules among the modules in $\mathbf{Ns}_r \cup \{0_t\})$ is at least $T_r + 2$.
Because the t-system consists of only one module and the r-system is supposed to contain at most T_r maliciously behaving modules, the number of maliciously behaving modules in $\mathbf{Ns}_r \cup \{0_t\})$ is at most $T_r + 1$.
Hence if both modules d and e are behaving correctly, the decisions $dec(d_r)$ and $dec(e_r)$ must be identical, i.e.:

$$\forall d_r, e_r\ :\ d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = dec(e_r) \qquad (5.2)$$

Which completes the proof of Theorem 5.1. □

5.2.2 The DJC Method applied to a fault-tolerant input device with post-observation

Suppose the input device is a fault-tolerant system. In this case too we do not want such an input device to cause the receiving system to go down, regardless of whether the fault-tolerant input system is functioning correctly or behaving maliciously.

Let the t-system consist of N_t modules and let each of these modules deliver one symbol of the encoded output. So module i_t delivers the partially encoded version $m(i_t)$ of some data value x_t, i.e. $m(i_t) = \mathcal{Y}t(i_t)(x_t)$, where $\mathcal{Y}t$ is a T_t-error-correcting code. The observing function is as usual denoted by $\mathcal{Y}t^{(-1)}$. Then the DJC Method applied to a fault-tolerant input device with post-observation is defined as follows:

Definition 5.3
Let the observing function of the t-system be denoted by $\mathcal{Y}t^{(-1)}$, *then*

- *All modules* i_t *in the t-system communicate their messages* $m(i_t) = \mathcal{Y}t(i_t)(x_t)$ *to all modules* d_r *of the r-system by means of DJC algorithms which are chosen from the classes* $\mathcal{A}(T_r, T_r + 2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$.
- *The decision* $dec(i_r)$ *in a module* d_r *of the r-system is the result of the observing function* $\mathcal{Y}t^{(-1)}$ *applied on the* N_t *decisions* $dec_{T_r+2}((i_t), d_r)$ *of the algorithms chosen from the classes* $\mathcal{A}(T_r, T_r+2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$.

□

The indication "post-observation" stems from the fact that the observing function is applied after the DJC algorithms are applied.

The method is shown in Figure 5.2. In this figure the algorithms chosen from the classes $\mathcal{A}(T_r, T_r + 2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$. are again decomposed in algorithms chosen from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ surrounded by the encoding function $\mathcal{Y}_{(i_t)}$ and the decoder function $\mathcal{Y}_{(i_t)}^{(-1)}$. The communications due to the algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ are denoted by o o o.
Firstly, notice that the choice $\mathbf{B}(i_t) = \mathbf{B}(j_t)$ which is suggested by Figure 5.2 is not required. The sets $\mathbf{B}(i_t)$ only have to satisfy the definition of the DJC algorithms, thus $\mathbf{B}(i_t) \subset \mathbf{Ns}_r$ and $|\mathbf{B}(i_t)| \geq 2T_r + 1$.
Secondly, notice that the algorithms from the classes $\mathcal{A}(T_r, T_r+1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ are again IAC algorithms.

In each module d_r the algorithms chosen from the classes $\mathcal{A}(T_r, T_r + 2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$, with $i_t \in \mathbf{Ns}_t$ result in N_t decisions $dec_{T_r+2}((i_t), d_r)$. Ignoring possible faults, these decisions will be equal to the partial encoded values of x_t. So the value x_t is calculated from the decisions $dec_{T_r+2}((i_t), d_r)$ by means of the observing function $\mathcal{Y}t^{(-1)}$.

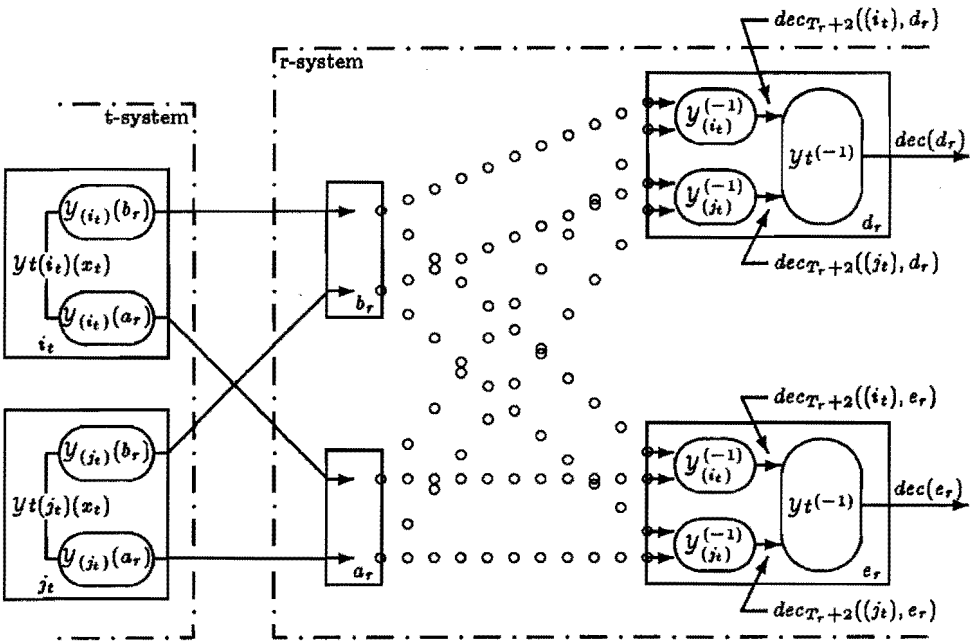


Figure 5.2: *The DJC Method applied to a fault-tolerant input device with post-observation*

From Theorem 3.1 we know that the class $\mathcal{A}(T_r, T_r+2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$ is non-empty if and only if $|\mathbf{Ns}_r \cup \{i_t\}| \geq 2T_r + T_r + 2$. Hence our algorithm always can be implemented if $N_r \geq 3T_r + 1$.

A fault-tolerant input device which is connected to a fault-tolerant system according to the method defined above can never cause the latter system to go down, regardless of whether the fault-tolerant input device functions correctly or not, i.e.:

Theorem 5.2 *If the DJC Method with post-observation is applied for connecting a fault-tolerant input device, the t-system, to a fault-tolerant system, the r-system, as is defined in Definition 5.3, then the t-system and the r-system satisfy the interactive consistency requirement between communicating systems as is defined in Definition 5.1.* □

Proof:

From Definition 5.3 it follows that a DJC method applied to a fault-tolerant input device with post-observation is constructed from an N_t-fold application of the DJC method for a single input device, followed by the application of the observing function in each module of the r-system.

From Theorem 5.1 we know that when a message $m(i_t)$ is communicated by module i_t of the t-system to the modules d_r of the r-system by means of algorithms from the classes $\mathcal{A}(T_r, T_r+2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$ the following holds:
1. If the module i_t is functioning correctly, then the decision $dec_{T_r+2}((i_t), d_r)$ of the algorithm from the class $\mathcal{A}(T_r, T_r+2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$ which is calculated in a correctly functioning module d_r, equals the message $m(i_t)$.
2. Regardless of the correctness of the module i_t, any pair of decisions calculated in correctly functioning modules d_r and e_r satisfies $dec_{T_r+2}((i_t), d_r) = dec_{T_r+2}((i_t), e_r)$.

From the preceding it follows that after the DJC algorithms have been completed:
1. If the t-system is functioning correctly, i.e. at least $N_t - T_t$ modules forwarded a message $m(i_t) = \mathcal{Y}t(i_t)(x_t)$, then in a correct module d_r of the r-system at least for $N_t - T_t$ decisions holds

$$dec_{T_r+2}((i_t), d_r) = m(i_t) = \mathcal{Y}t(i_t)(x_t)$$

And thus if the observing function $\mathcal{Y}t^{(-1)}$ is applied on these values $dec_{T_r+2}((i_t), d_r)$ then the result will be

$$dec(d_r) = x_t$$

2. Regardless of whether the t-system is functioning correctly or not, in all correctly functioning modules d_r, the same set of N_t decisions $dec_{T_r+2}((i_t), d_r)$ is available and in all modules the same observing function $\mathcal{Y}t^{(-1)}$ is applied on these values. So the results $dec(d_r)$ will be the same in all correctly functioning modules.

The fact that d_r is functioning correctly is again expressed by $d_r \in \overline{\mathbf{F}}_r$, and the fact the the t-system is functioning correctly by $|\overline{\mathbf{F}}_t| \geq N_t - T_t$. So from the preceding we obtain

$$\forall \overline{\mathbf{F}}_t, \overline{\mathbf{F}}_r \ : \ |\overline{\mathbf{F}}_r| \geq N_r - T_r \Longrightarrow$$
$$\Big(\big(|\overline{\mathbf{F}}_t| \geq N_t - T_t \Longrightarrow [\ \forall d_r \ : \ d_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = x_t \ ] \big) \wedge$$
$$[\ \forall d_r, e_r \ : \ d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = dec(e_r) \ ] \Big) \tag{5.3}$$

Which completes the proof of Theorem 5.2. □

5.2.3 The DJC Method applied to a fault-tolerant input device with pre-observation

The pre-observation method is an efficient alternative to the post-observation method. Again we start from the assumption that the input device is a fault-tolerant system of which the output is encoded by means of an observing function $\mathcal{Y}t$ able to correct T_t errors.
The t-system consists of N_t modules and each of these modules delivers one symbol of the encoded output. So module i_t delivers the partially encoded version $m(i_t)$ of a data value x_t. I.e. $m(i_t) = \mathcal{Y}t(i_t)(x_t)$. The observation function is again denoted by $\mathcal{Y}t^{(-1)}$. Then the DJC Method applied to a fault-tolerant input device with pre-observation is defined as follows:

Definition 5.4
Let **Inp** *be a set of N_{inp} modules in the r-system which provide for the communication from the t-system, such that $2T_r + 1 \leq N_{inp} \leq N_r$.*

- *Each modules i_t in the t-system sends its messages $m(i_t)$, with $m(i_t) = \mathcal{Y}t(i_t)(x_t)$, to the N_{inp} modules of the set* **Inp** *in the r-system.*
- *In each module a_r of the r-system with $a_r \in$* **Inp**, *the observing function $\mathcal{Y}t(i_t)$, is applied on the N_t messages received from the t-system. Let the result obtained in module a_r be denoted by $m(a_r)$.*
- *In module a_r the message $m(a_r)$ is encoded by means of the partial encoder function $\mathcal{W}(a_r)$ of a T_r-error-correcting code $\mathcal{W}$ of which the number of symbols in a code word is N_{inp}. Let the N_{inp} results be denoted by the function y, hence $y(a_r) = \mathcal{W}(a_r)(m(a_r))$.*
- *Each module a_r communicates its message $y(a_r)$ to all modules of the r-system by means of a DJC algorithm (is an IAC algorithm) from the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$.*

- *The decision* $dec(d_r)$ *in a module* d_r *of the r-system is the result of the decoding function* $\mathcal{W}^{(-1)}$ *applied on the* N_{inp} *decisions* $dec_{T_r+1}((a_r), d_r)$ *of the IAC algorithms which are chosen from the classes* $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$.

□

The indication "pre-observation" stems from the fact that the observing function of the t-system is applied before the DJC algorithms.

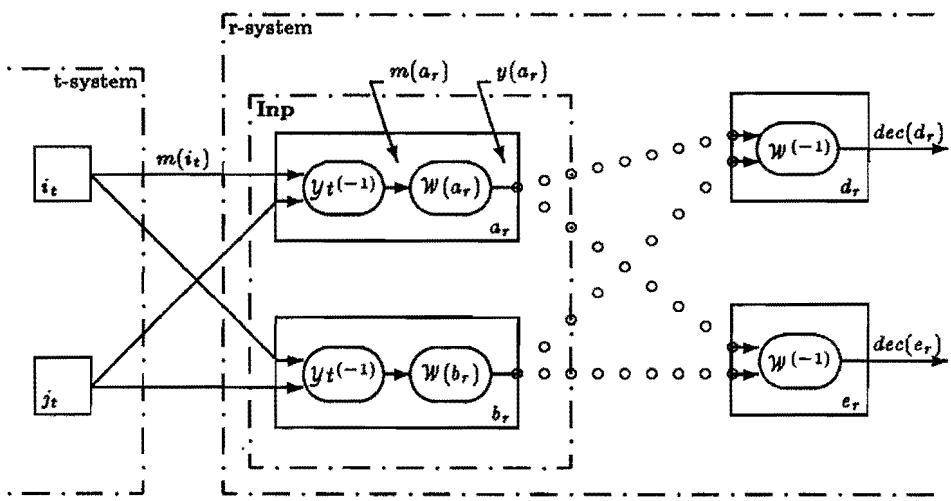


Figure 5.3: *The DJC Method applied to a fault-tolerant input device with pre-observation*

The method is shown in Figure 5.3. In this figure the communications due to the algorithms which are chosen from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ are denoted by o o o.
In the modules a_r with $a_r \in \mathbf{Inp}$ an estimate is calculated for the value x_t. Ignoring possible faults, these estimates $m(a_r)$ will be equal to x_t. The estimates are partially encoded like in an (N, K)-concept and forwarded to all modules d_r of the r-system by means of DJC algorithms from the class $\mathcal{A}(T_r, T_r+1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ with $a_r \in \mathbf{Inp}$. In each module d_r these algorithms will result in N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$. Again ignoring possible faults, these decisions will be equal to the partial encoded values of x_t. So the value

x_t is calculated from the N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$ by means of the decoding function $\mathcal{W}^{(-1)}$.

From the definition it follows that after the observing function $\mathcal{Y}t$ is applied, an Interactive Consistency algorithm is applied for each message $y(a_r)$. So, the method can clearly always be applied if the r-system contains at least $3T_r + 1$ modules. In that case it is obviously possible to satisfy the condition $2T_r + 1 \leq N_{inp} \leq 3T_r + 1 \leq N_r$ from Definition 5.4.

A fault-tolerant input device which is connected to a fault-tolerant system according to the method defined above can never cause the latter system to go down, regardless of whether the fault-tolerant input device functions correctly or not, i.e.:

Theorem 5.3 *If the DJC Method with pre-observation is used for connecting a fault-tolerant input device, the t-system, to a fault-tolerant system, the r-system, as is defined in Definition 5.4, then the t-system and the r-system satisfy the interactive consistency requirement between communicating systems as is defined in Definition 5.1.* □

Proof:

Let the set of correctly functioning modules in the t-system and the r-system again be denoted by $\overline{\mathbf{F}}_t$ and $\overline{\mathbf{F}}_r$ respectively. Recall that $|\overline{\mathbf{F}}_t| \geq N_t - T_t$ indicates that the t-system is functioning correctly.
If the t-system is functioning correctly then the result $m(a_r)$ of the observing function $\mathcal{Y}t^{(-1)}$ applied in a correctly functioning module a_r on the received messages $m(i_t)$ will be equal to the original message value x_t in the t-system, i.e.:

$$\left(|\overline{\mathbf{F}}_t| \geq N_t - T_t \wedge a_r \in \mathbf{Inp} \wedge a_r \in \overline{\mathbf{F}}_r\right) \Longrightarrow m(a_r) = x_t \qquad (5.4)$$

According to Definition 5.4 the results $m(a_r)$ are encoded by means of the partial encoder function $\mathcal{W}(a_r)$ of a T_r-error-correcting code such that $y(a_r) = \mathcal{W}(a_r)(m(a_r))$. And thus if both the t-system and module a_r are functioning correctly it holds that $y(a_r) = \mathcal{W}(a_r)(x_t)$.

The values $y(a_r)$ with $a_r \in \mathbf{Inp}$ are communicated by Interactive Consistency algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ to all modules of the r-system. So for the results $dec_{T_r+1}((a_r), d_r)$ of these IAC algorithm it holds

that

$$a_r, d_r \in \overline{\mathbf{F}}_r \Longrightarrow dec_{T_r+1}((a_r), d_r) = y(a_r)$$
$$d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec_{T_r+1}((a_r), d_r) = dec_{T_r+1}((a_r), e_r) \tag{5.5}$$

After the IAC algorithms have been completed, each module d_r in the r-system contains N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$. If both the t-system and module a_r are functioning correctly it thus holds that $dec_{T_r+1}((a_r), d_r) = \mathcal{W}(a_r)(x_t)$. In each module d_r the decoder function $\mathcal{W}^{(-1)}$ of the T_r-error-correcting code is applied to these decisions $dec_{T_r+1}((a_r), d_r)$. The result is denoted by $dec(d_r)$. At most T_r modules a_r are behaving maliciously. Hence if the t-system is functioning correctly, then $dec(d_r) = x_r$. So

$$\left(|\overline{\mathbf{F}}_t| \geq N_t - T_t \wedge d_r \in \overline{\mathbf{F}}_r\right) \Longrightarrow dec(d_r) = x_t \tag{5.6}$$

which proves the first property of interactive consistency between communicating fault-tolerant systems.

From the properties (5.5) of the IAC algorithm we know that all correctly functioning modules arrive at the same decision $dec_{T_r+1}((a_r), d_r)$, regardless of whether the module a_r is functioning correctly or maliciously. In all modules d_r the same decoder function is applied to the values $dec_{T_r+1}((a_r), d_r)$. Hence the results $dec(d_r)$ must be the same in all modules. I.e

$$d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = dec(e_r) \tag{5.7}$$

which proves the second property of interactive consistency between communicating fault-tolerant systems. □

The DJC method with pre-observation will generally require less communication than the DJC method with post-observation. This can be explained as follows:
Suppose the sets $\mathbf{B}(i_t)$ in the method based on post-observation and the set **Inp** in the method based on pre-observation are of the same size. Let this size be N''.
In both methods during round 0 the partially encoded messages $\mathcal{Y}t(i_t)(x_t)$ which are available in the modules i_t of the t-system are each transmitted

to the N'' modules of the r-system. However, in the DJC method with post-observation they are first encoded by means of the encoding function $\mathcal{Y}_{(i_t)}$ of the algorithm from the class $\mathcal{A}(T_r, T_r+2, i_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{i_t\})$. So in the case of post-observation the size of the messages transmitted during round 0 will generally be smaller and thus the number of messages counted according to the message size will be smaller too.
However, in the case of post-observation all $N_t.N''$ messages which are received at the end of round 0 by the modules of the r-system, have to be communicated to all modules of the r-system by means of an algorithm from the classes $\mathcal{A}(T_r, T_r+1, a_t, \mathbf{Ns}_r, \mathbf{Ns}_r)$, whereas in the case of pre-observation only N'' messages have to be communicated to all modules of the r-system by algorithms from the same classes. The messages which are to be communicated in round 1 are in both cases partially encoded copies of the message $\mathcal{Y}t(i_t)(x_t)$. In case of post-observation the encoding function is $\mathcal{Y}_{(i_t)}$ and in the case of pre-observation the encoding function is $\mathcal{W}(a_r)$. Because the size reduction due to these encoding functions will be about the same, the number of messages which are to be transmitted after rounds 0 will in the case of pre-observation be a factor N'' smaller compared to post-observation. This reduction in general counts more than the fact that during round 0 the number of messages in the case of pre-observation is more than in the case of post-observation.

5.2.4 The DJC Method applied to an NMR input device with pre-coding and pre-observation

Though pre-observation is an efficient alternative to the post-observation method, further improvements can be obtained if the t-system can be considered as an NMR system, i.e. if the observing function of the t-system is a repetition code. In that case the partial encoder functions $\mathcal{W}(a_r)$ may be moved backwards in the data flow from the modules in the r-system to the modules in the t-system. This causes a reduction in the amount of information which is to be transmitted between the t-system and the r-system.

The DJC Method applied to an NMR input device with pre-observation and pre-coding is defined as follows:

Definition 5.5
Let the t-system be an NMR fault-tolerant input device and let **Inp** *be a set*

of N_{inp} modules in the r-system which provides for the communication from the t-system, such that $2T_r + 1 \leq N_{inp} \leq N_r$.

- *Each correctly functioning module i_t in the t-system possesses the original message value x_t. In each module i_t this value is encoded by means of a T_r-error-correcting code $\mathcal{W}$ in which a code word consists of N_{inp} symbols. The N_{inp} symbols are each sent to a different module of the set* **Inp**. *So the partial encoded message $w(i_t, a_r) = \mathcal{W}(a_r)(x_t)$ is sent from module i_t in the t-system to module a_r in the set of modules* **Inp** *in the r-system.*
- *In each module a_r of the r-system with $a_r \in$* **Inp**, *a majority vote Maj is applied to the N_t messages $w(i_t, a_r)$ received from the t-system. Let the result obtained in module a_r be denoted by $m(a_r)$.*
- *Each module a_r with $a_r \in$* **Inp** *communicates its message $m(a_r)$ to all modules of the r-system by means of a DJC algorithm from the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ (this is an IAC algorithm).*
- *The decision $dec(d_r)$ in a module d_r of the r-system is the result of the decoding function $\mathcal{W}^{(-1)}$ applied to the N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$ with $a_r \in$* **Inp** *resulting from the IAC algorithms.*

The method is elucidated in Figure 5.4. In this figure the IAC algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ with $a_r \in \mathbf{Inp}$ are again denoted by o o o.
In the modules a_r with $a_r \in \mathbf{Inp}$ an estimate is calculated of the value $\mathcal{W}(a_r)(x_t)$. These estimates are forwarded to all modules d_r of the r-system by means of DJC algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$. In each module d_r these IAC algorithms result in N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$. Ignoring possible faults, these decisions will be equal to the partial encoded values $\mathcal{W}(a_r)(x_t)$. So the value x_t is calculated from the decisions $dec_{T_r+1}((a_r), d_r)$ by means of the decoding function $\mathcal{W}^{(-1)}$.

Although the observing function $\mathcal{Y}t^{(-1)}$ of the NMR t-system is a majority vote, this function differs from the function Maj applied in the modules a_r in the set **Inp**, because $\mathcal{Y}t^{(-1)}$ operates on the original message x_t and Maj operates on the partially encoded versions of x_t.

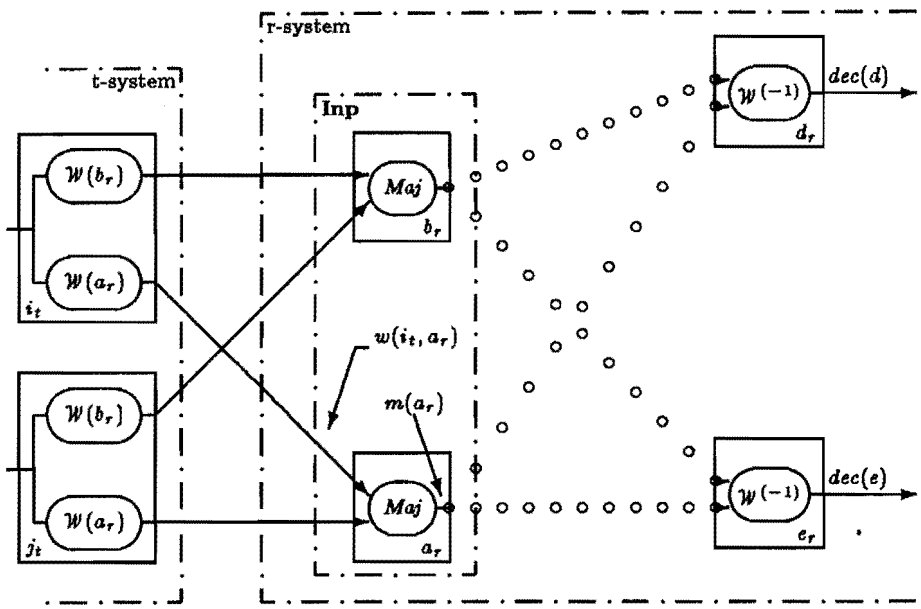


Figure 5.4: *The DJC Method applied to an NMR fault-tolerant input device with post-observation and pre-coding*

From the definition we know that after the majority vote Maj has been applied in the modules a_r of the r-system which provide for the input, an Interactive Consistency algorithm is applied for each module a_r. So clearly the method can always be implemented if the r-system contains at least $3T_r + 1$ modules.

An NMR fault-tolerant input device which is connected to a fault-tolerant system according to the method defined above can never cause the latter system to go down, regardless whether the fault-tolerant input device is functioning correctly or not, i.e.:

Theorem 5.4 *If the DJC Method with pre-observation and pre-coding is applied for connecting an NMR fault-tolerant input device, the t-system, to a fault-tolerant system, the r-system, as is defined in Definition 5.5, then the t-system and the r-system satisfy the interactive consistency requirement between communicating systems as is defined in Definition 5.1.* □

Proof:

Let the set of correctly functioning modules in the t-system and the r-system again be denoted by $\overline{\mathbf{F}}_t$ and $\overline{\mathbf{F}}_r$ respectively. Recall that $|\overline{\mathbf{F}}_t| \geq N_t - T_t$ indicates that the t-system is functioning correctly. Because the t-system is an NMR system it holds that $N_t \geq 2T_t + 1$.

If the t-system is functioning correctly then each correctly functioning module a_r in the set **Inp** will receive at least $T_t + 1$ values $w(i_t, a_r)$ such that $w(i_t, a_r) = \mathcal{W}(a_r)(x_t)$. Hence the result of the majority vote will be $m(a_r) = \mathcal{W}(a_r)(x_t)$. I.e.

$$\left(|\overline{\mathbf{F}}_t| \geq N_t - T_t \wedge a_r \in \mathbf{Inp} \wedge a_r \in \overline{\mathbf{F}}_r\right) \Longrightarrow m(a_r) = \mathcal{W}(a_r)(x_t) \tag{5.8}$$

The values $m(a_r)$ are communicated by means of Interactive Consistency algorithms to all modules of the r-system. So for the results $dec_{T_r+1}((a_r), d_r)$ of the algorithms chosen from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ it holds that

$$\begin{aligned} a_r, d_r \in \overline{\mathbf{F}}_r &\Longrightarrow dec_{T_r+1}((a_r), d_r) = m(a_r) \\ d_r, e_r \in \overline{\mathbf{F}}_r &\Longrightarrow dec_{T_r+1}((a_r), d_r) = dec_{T_r+1}((a_r), e_r) \end{aligned} \tag{5.9}$$

After the IAC algorithm has been completed each module d_r in the r-system contains N_{inp} decisions $dec_{T_r+1}((a_r), d_r)$. If both the t-system and module a_r are functioning correctly it thus holds that $dec_{T_r+1}((a_r), d_r) = \mathcal{W}(a_r)(x_r)$. On these decisions $dec_{T_r+1}((a_r), d_r)$ in each module d_r the decoder function $\mathcal{W}^{(-1)}$ of the T_r-error-correcting code is applied. The result is denoted by $dec(d_r)$. At most T_r modules a_r are behaving maliciously. Hence if the t-system is functioning correctly, then $dec(d_r) = x_r$. So

$$\left(|\overline{\mathbf{F}}_t| \geq N_t - T_t \wedge d_r \in \overline{\mathbf{F}}_r\right) \Longrightarrow dec(d_r) = x_t \tag{5.10}$$

which proves the first property of interactive consistency between communicating fault-tolerant systems.

From the properties (5.9) of the IAC algorithm we know that all correctly functioning modules arrive at the same decision $dec_{T_r+1}((a_r), d_r)$, regardless

of whether the module a_r is functioning correctly or maliciously. In all modules d_r the same decoder function is applied on the values $dec_{T_r+1}((a_r), d_r)$. Hence the results $dec(d_r)$ must be the same in all modules. I.e

$$d_r, e_r \in \overline{\mathbf{F}}_r \Longrightarrow dec(d_r) = dec(e_r) \tag{5.11}$$

which proves the second property of interactive consistency between communicating fault-tolerant systems. □

5.3 Some examples of the interconnection of fault-tolerant systems

In this section we will elaborate on some system parameters imposed by the requirement of interactive consistency between communicating fault-tolerant systems. Moreover we will present a number of simple examples of the interconnection of fault-tolerant systems with $T_t = 0$ or 1, and $T_r = 1$. For these examples the number of messages which needs to be transmitted in order to obtain interactive consistency between communicating systems will be presented. Finally, the I/O architecture of a $(4, 2)$ concept fault-tolerant system will be discussed.

5.3.1 An (N, K)-concept fault-tolerant system interconnected with external sources

In the previous section it was shown that whenever it is required that a fault-tolerant system which is based on generalized masking must not be defeated by a malfunctioning source then it is necessary that N_r is greater than $3T_r$.

For an (N, K)-concept fault-tolerant computer, in which a T-error-correcting MDS code is applied, it holds that $N - K = 2T$. The requirement of interactive consistency between communicating systems imposes $N \geq 3T + 1$. Hence, choosing the minimum value for N, i.e. $3T + 1$, we get $K = T + 1$. So we obtain an $(N, K) = (3T + 1, T + 1)$-concept fault-tolerant computer which is able to tolerate T random module failures simultaneously and which guarantees interactive consistency in the fault-free modules when a malfunctioning source is connected to it.

In the (N, K)-concept fault-tolerant computer the amount of processor hardware is N fold compared with a non-redundant system and the amount of

memory hardware needed is N/K times the amount of memory hardware needed in a non-redundant system.
In the $(3T+1, T+1)$-concept fault-tolerant computer the amount of processor hardware is determined by the interactive consistency requirement. With respect to the non-redundant system the amount of memory hardware is only duplicated for $T = 1$ and almost triplicated for large T. This is a considerable improvement compared to an N-modular redundant system in which the amount of processor hardware and also the amount of memory hardware are $3T + 1$ times as much as in a non-redundant system.
Thus the (N, K)-concept generalization makes it possible to adapt the masking redundancy requirements to the interactive consistency requirements.

5.3.2 Some simple examples of the interconnection of fault-tolerant systems

Suppose we are dealing with a $(4, 2)$-concept receiving system and a single input device. Thus

$$N_t = 1 \text{ and } T_t = 0 \text{ and } N_r = 4 \text{ and } T_r = 1$$

Let the single module in the t-system again be denoted by 0_t. When using the DJC method applied to a single input device, we know from Definition 5.2 that the data value x_t in the input device 0_t is communicated by means of algorithms from the class $\mathcal{A}(1, 3, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$ to the modules d_r in $\mathbf{Ns}_r$. The next-set $\mathbf{B}(0_t)$ must contain at least 3 modules, because the code is to be single error-correcting, but can contain at most 4 modules because $\mathbf{B}(0_t) \subset \mathbf{Ns}_r$. Similarly the next-sets $\mathbf{B}(0_t, a_r)$ must contain at least 3 modules. From the elucidation to the construction of the DJC algorithms on page 153, equation (3.38) we know $\mathbf{B}(0_t, a_r) \subset (\mathbf{Ns}_r - \{a_r\})$. Hence $|\mathbf{B}(0_t, a_r)| = 3$.
So for the code $\mathcal{Y}_{(0_t)}$ used by the algorithm from the class $\mathcal{A}(1, 3, 0_t, \mathbf{Ns}_r, \mathbf{Ns}_r \cup \{0_t\})$ we may choose between a $[4, 2, 2]$ code and a $[3, 1, 1]$ code. Recall from Chapter 4 that a $[n_c, k_c, b_c]$ code indicates a code consisting of code words of n_c symbols, data words of k_c symbols and symbols which are represented by b_c bits.
The codes $\mathcal{Y}_{(0_t, a_r)}$ used by the algorithms from the classes $\mathcal{A}(1, 3, a_r, \mathbf{Ns}_r, \mathbf{Ns}_r)$ must consist of 3-symbol code words. So if $\mathcal{Y}_{(0_t)}$ is a $[4, 2, 2]$ code then $\mathcal{Y}_{(0_t, a_r)}$ must be a $[3, 1, 2]$ code, and if $\mathcal{Y}_{(0_t)}$ is a $[3, 1, 1]$ code then $\mathcal{Y}_{(0_t, a_r)}$ must be a $[3, 1, 1]$ code.

The implementation of the method will be characterized by the encoder and decoder functions applied in the order of the data flow.
We will use the following notation:

$[n_c, k_c, b_c]$	Denotes the application of the encoding function of a $[n_c, k_c, b_c]$ error-correcting code. The n_c results are sent in n_c different directions.
$\langle n_c, k_c, b_c \rangle$	Denotes the application of the decoding function of a $[n_c, k_c, b_c]$ error-correcting code.
t	Indicates that the data is transmitted.
–	Indicates that the data will be sent unchanged to all modules in the set of destinations which has not yet been passed by the message.
s	Indicates that one of the symbols of the code word is selected for transmission.

For example in one of the possible implementations of the DJC method applied to a single input device which was mentioned above, the $[3,1,1]$ code is applied twice.
This example is characterized as follows:

	Before round 0 of the DJC algorithm:
$[1,1,1]$	First the observing function yt of the single input device is denoted by a 0-error-correcting code $[1,1,1]$.
	During round 0, first part, of the DJC algorithm:
$[3,1,1]$	Then the encoder function $y_{(0_t)}$ is applied, which is a $[3,1,1]$ code.
	During round 0, second part:
t	The data is transmitted to the r-system, which is indicated by t.
	During round 1, first part:
$[3,1,1]$	Again a $[3,1,1]$ code is applied, i.e. the encoder functions $y_{(0_t,a_r)}$.
	During round 1, second part:
t	And the data is again sent to the next modules, which is indicated by t.
	During round 2:

$-t$ During the last round the messages are broadcast to all destinations which have not yet been passed by the messages. The broadcasting is indicated by '$-$' and the transmission by t.

During round 3, first part:

$\langle 3,1,1\rangle$ In the destination first the decoder functions $\mathcal{Y}^{(-1)}_{(0_t,a_r)}$ of the $[3,1,1]$ code are executed; this is indicated by $\langle 3,1,1\rangle$.

During round 3, second part:

$\langle 3,1,1\rangle$ Thereafter the decoder functions $\mathcal{Y}^{(-1)}_{(0_t)}$ code are executed.

After round 3 of the DJC algorithm:

$\langle 1,1,1\rangle$ And finally the observation function denoted by $\langle 1,1,1\rangle$ is applied. Notice that the observation function in this case is just an identity function.

So our example is characterized by:

$$[1,1,1][3,1,1]t[3,1,1]t - t\langle 3,1,1\rangle\langle 3,1,1\rangle\langle 1,1,1\rangle$$

In the same way the other implementation is characterized by

$$[1,1,4][4,2,2]t[3,1,2]t - t\langle 3,1,2\rangle\langle 4,2,2\rangle\langle 1,1,4\rangle$$

The number of messages which needs to be transmitted and the minimum size of the original message x_t in the "source" can be easily calculated from this notation.
For example in our second example we start from messages of 4 bits. These are encoded into 4 symbols of 2 bits and each symbol is transmitted during round 0. This gives $4 \times 0.5 = 2$ unit messages. During round 1, each symbol is encoded with a $[3,1,2]$ code, resulting in $3 \times 4 = 12$ messages of 2 bits, i.e. 6 unit messages. In the last round each two-bit message is sent to the remaining 2 destinations causing 24 messages of 2 of unit messages which needs to be transmitted in this algorithm is 20.

In the same way the other methods can be characterized by the succession of coders, transmissions and decoders.
If in a module only one partial encoder function is applied, like the function $\mathcal{W}(a_r)$ in the DJC method with pre-observation, this will be denoted by an s following the code, indicating that only one symbol of the encoding is selected for transmission.

In Table 5.1 a number of implementations are presented.

The DJC Method applied to a single source $T_t = 0, \quad N_r = 4, \quad T_r = 1$					
N_t	code sequence	m.size	#mess t-> r	#mess r-> r	#mess tot.
1	[1,1,1][3,1,1]t[3,1,1]t-t⟨3,1,1⟩⟨3,1,1⟩⟨1,1,1⟩	1	3	27	30
1	[1,1,4][4,2,2]t[3,1,2]t-t⟨3,1,2⟩⟨4,2,2⟩⟨1,1,4⟩	4	2	18	20

The DJC Method applied to a fault-tolerant source with post-observation $T_t = 1, \quad N_r = 4, \quad T_r = 1$					
N_t	code sequence	m.size	#mess t-> r	#mess r-> r	#mess tot.
3	[3,1,1][3,1,1]t[3,1,1]t-t⟨3,1,1⟩⟨3,1,1⟩⟨3,1,1⟩	1	9	81	90
3	[3,1,4][4,2,2]t[3,1,1]t-t⟨3,1,1⟩⟨4,2,2⟩⟨3,1,4⟩	4	6	54	60
4	[4,2,2][3,1,2]t[3,1,2]t-t⟨3,1,2⟩⟨3,1,2⟩⟨4,2,2⟩	4	6	54	60
4	[4,2,4][4,2,2]t[3,1,2]t-t⟨3,1,2⟩⟨4,2,2⟩⟨4,2,4⟩	8	4	36	40

The DJC Method applied to a fault-tolerant source with pre-observation $T_t = 1, \quad N_r = 4, \quad T_r = 1$					
N_t	code sequence	m.size	#mess t-> r	#mess r-> r	#mess tot.
3	[3,1,1][3,1,1]t⟨3,1,1 ⟩[1,1,1]s[3,1,1]t-t⟨3,1,1⟩⟨3,1,1⟩	1	9	27	36
3	[3,1,4][4,1,4]t⟨3,1,4 ⟩[4,2,2]s[3,1,2]t-t⟨3,1,2⟩⟨4,2,2⟩	4	12	18	30
4	[4,2,2][3,1,2]t⟨4,2,2 ⟩[1,1,4]s[3,1,4]t-t⟨3,1,4⟩⟨3,1,4⟩	4	6	27	33
4	[4,2,2][4,1,2]t⟨4,2,2 ⟩[4,2,2]s[3,1,2]t-t⟨3,1,2⟩⟨4,2,2⟩	4	8	18	26

The DJC Method applied with pre-observation and pre-coding $T_t = 1, \quad N_r = 4, \quad T_r = 1$					
N_t	code sequence	m.size	#mess t-> r	#mess r-> r	#mess tot.
3	[3,1,4][4,2,2]t⟨3,1,2 ⟩[3,1,2]t-t⟨3,1,2⟩⟨4,2,2⟩	4	6	18	24

Table 5.1: *A comparison of the four different DJC Methods for obtaining interactive consistency between communicating modules.*

5.3.3 The architecture of a (4,2) module

In order to implement the preceding algorithms each module of the (4,2) concept is provided with a separate I/O processor or I/O controller. In Figure 5.5 the architecture of one module is shown. The I/O processor I/OPROC has its own random access memory I/OMEM. This memory cannot be accessed by the main processor MAINPROC, so communication between the main processor and the I/O processor takes place via the main memory MAINMEM. A separate I/O processor is chosen because in most multiprocessor systems the algorithms and protocols needed for the I/O are a considerable load for the system. When the algorithms described in the preceding section are implemented the load is even larger.

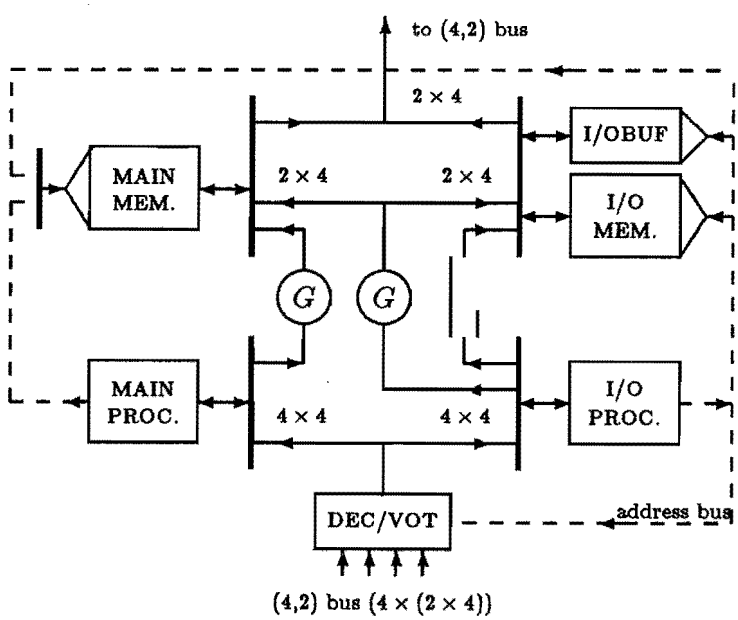


Figure 5.5: The architecture of the (4,2)-concept provided with the means for reaching interactive consistency

The decoders DEC/VOT and the interconnection bus between the modules, the (4,2) bus, are shared by the main processor and the I/O processor.

The decoders not only act as a decoder for the (4,2) code but they also take care of the broadcasting and voting functions needed for the interactive consistency algorithm.

Messages received from outside sources are stored in I/O buffers I/OBUF, which are connected to the I/O memory bus (memory mapped I/O). An interrupt is treated like a message. The I/O processors poll these I/O buffers.

The easiest way to explain the architecture shown in Figure (5.5) is to describe the data transfers in the module caused by reading one of the I/O buffers in one of the modules.

Suppose an I/O buffer in module 0 has to be read, then all four I/O processors perform synchronously a read operation on their I/O. Only the address on the address bus of the I/O processor in module 0 is able to select an I/O buffer. The data from the selected I/O buffer in module 0 is transferred over the 8-bit wide I/O bus to the drivers of the (4,2) bus. The other modules transfer zero information via their I/O buses and drivers to the (4,2) bus. At the same time the address bus of the I/O processor forces the decoder into a mode called the broadcast mode, in which the 8 bits from module 0 are transferred to the lower byte output of the decoder. This is done in all modules, so the data byte from the I/O buffer in module 0 is now transferred to all I/O processors.

The next step of the algorithm is to write the byte in a memory location (of I/OMEM.) specially reserved for data descending from the I/O buffers of module 0. This is done by a synchronous write operation in the four modules. The encoders are bypassed during this operation, so the bytes are unchanged.

In the following step a read operation is performed on the memory location in which the byte was stored. This is done synchronously by all four processors. So in the fault-free case four identical bytes appear at the (4,2) bus in parallel. Again the address bus of the I/O processor forces the decoders into another special mode, called the voting mode, in which the decoders take a majority decision on the bytes received from the modules 1, 2 and 3 (the byte received from module 0 is neglected). The reserved address for data descending from the I/O buffers of module 0 is of course the same one that forces the decoder into this special mode. After this read operation all non-faulty modules contain identical data, regardless of whether module 0 was

producing broadcast errors or one of the other modules was malfunctioning. The four bytes residing in the four I/O processors can now be treated as general data in the $(4,2)$-concept. Thus during the next step the data can be written into the main memory, during which write operation the four bytes are encoded.

In order to cope with a single source, this procedure must be executed for each of the three copies of the message sent by the single source to the modules of the (4,2)-concept and a majority vote has to be taken on the results.

5.3.4 Some concluding remarks

The (N,K)-concept makes it possible to choose the ratio between memory and processor redundancy. This ratio can be chosen such that the total amount of hardware is minimal. However it is often more important to reduce the number of service calls. In that case the added redundancy for the most unreliable components should be as small as possible. At first glance this contradicts the reliability requirements. The reliability improvement however depends on the number of failing modules that can be tolerated. In the (N,K) concept this is $(N-K)/2$. The added redundancy of the memory is $(N-K)/K$ and the added processor redundancy is $N-1$. So without influencing the reliability improvement, the number of service calls can be made minimal by choosing appropriate values for N and K. It should be noted that the bit-error-correcting property has to be taken into account. Furthermore it was shown that the (N,K) concept enables the number of modules to be adapted to the number that is needed to fulfil the interactive consistency requirements.

Chapter 6

Conclusions

In this thesis we have shown that methods for improving the reliability of digital systems which are based on N-modular redundancy can be generalized to methods which are based on a distributed implementation of error-correcting codes. These methods have been called "Generalized Masking Redundancy"

Within the class of fault-tolerant systems based on generalized masking two sub-classes can be identified which characterize the systems that are based on generalized masking.
The first class requires, after repair, some state initialization from the environment and is identified by the a distributing function $\mathcal{X}$, (the encoder function of an error-correcting code), an observing function $\mathcal{Y}$ (the decoder function of an error-correcting code) and the number T of maliciously behaving modules which are tolerated.
The second class re-initializes itself each time instance and is identified by a distributing function $\mathcal{X}$, an observing function $\mathcal{Y}$, a state decoder $\mathcal{Z}$ (the decoder function of an error-correcting code) and the number T of maliciously behaving modules which are tolerated.

It has been shown that in practical cases none of the two classes will suffice. Firstly, because re-initialization by external means causes the reliability of the system to depend on the environment. And secondly, because re-initializing the entire state each time instance is too costly. So practical implementations will be a mixture of these two classes, i.e. the system will re-initialize itself within a particular span of time.

An example of an architecture based on generalized masking is called the (N, K)-concept, of which the feasibility is proved by application in a commercial system. Due to the concept of generalized masking on which the (N, K)-concept is based the following is gained

- The ratio between memory and processor redundancy can be optimized with respect to the total amount of hardware needed or with respect to the call-rate of the system.
- Codes can be introduced which are capable of correcting both symbol and bit faults without requiring additional redundancy. In this way the code which is usually applied for memory protection is saved.
- The number of modules (fault isolation areas) required for fault tolerance can be adapted to the number of modules required for consistent communication with the environment.

The existence of codes which are able to correct both symbol and bit errors without requiring additional redundancy is shown with the presentation of such a symbol and bit-error-correcting code for the $(4, 2)$-concept. Moreover we have shown that a one-chip decoder for this code can be designed with a propagation delay of about 100nsec.

One of the worst problems in fault tolerance is the Input Problem. If special precautions are not taken, a correctly functioning fault-tolerant system might go down due to external faults. We have shown that this Input Problem is similar to the Interactive Consistency problem.
Interactive Consistency Algorithms give rise large numbers of messages which need to be exchanged between the modules. This limits their application to systems in which at most 3 or 4 faulty modules are tolerated.
In order to reduce the number of messages which need to be exchanged, a new class of Interactive Consistency Algorithms based on voting and coding has been defined. In practical applications in which less than 4 maliciously behaving modules are tolerated, these synchronous deterministic Interactive Consistency Algorithms turn out to be superior to their existing counterparts.

The properties of the class of Interactive Consistency Algorithms based on voting and coding have been proven on the basis of a newly defined class of

algorithms called Dispersed Joined Communication algorithms which have more liberal properties.

Finally, four algorithms based on Dispersed Joined Communication algorithms and Interactive Consistency algorithms have been presented which solve the Input Problem.

In this thesis we restricted ourselves to fault-tolerant systems in which the means for reliability improvement are implemented rather close to the hardware level. In general this turns out to be a cost-effective approach, both from the point of view of system cost and from the point of view of separation of concerns during the design process. It enables the development of software independently from the reliability requirements.
Implementation of the means for reliability improvement close to the hardware level naturally leads to a synchronous deterministic approach. This holds for the system fault tolerance as well as for the algorithms which solve the Input Problem.
However, if one wishes to start from off-the-shelf modules, the approach followed in this thesis will often not suffice. Firstly, a less strict definition of synchronism will be needed which is based on only time-outs. This will require adapted fault-tolerant synchronization algorithms. Secondly, the algorithms for solving the Input Problem will be applied on a higher level, in which case algorithms based on authentication will be cheaper. Much work in this "quasi asynchronous" field has still to be done.

The reliability of systems not only depends on the reliability of their constituting physical components but also on the correctness of the design. The field of specification and design description with respect to fault tolerance has hardly been explored. Only on the basis of formal description methods can proof systems be designed. Any result in this field will ease the validation problem of fault-tolerant systems.

Bibliography

[And. 79] T.Anderson, B.Randell, "The theory and practice of reliable system design", *Cambridge University Press*, 1979.

[And. 81] T.Anderson, P.A.Lee, "Fault tolerance principles and practice", *Prentice Hall International*, 1981

[Avizienis] A.Avizienis et al., "The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-tolerant Computer Design" *IEEE trans. Comp*, Vol.C20, No11, Nov.1971, pp.1312-1321.

[Baba 86] O.Babaoglu and R.Drumond, "Streets of Byzantium: network architectures for fast reliable broadcast" *IEEE Tr. on Softw. Eng.*, Vol.SE-11, No.6, 1986.

[Ben-Or 83] M.Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols" *Proc. 2nd ACM Symp. on Principles of Distributed Computing* Montreal, Canada, August 1983, pp.27-30.

[Bouricius] W.G.Bouricius, W.C.Carter, P.R. Schneider, "Reliability modeling techniques of selfrepairing computer systems", *Prc. of the 24th National Conf. of the ACM* pp295-333, 1969

[Boute 88] R.T.Boute, "System Semantics: Principles, Applications, and Implementation" *ACM Tr. on Progr. Lang. and Systems.* pp118-155, Vol.10, No.1, 1988.

[Bir 87] K.Birman and T.Joseph, "Reliable Communications in the Presence of Failures" *ACM Tr. on Comp. Syst.* Vol.5, No.1, 1987.

[Boly-88] J.P.Boly W.J. van Gils, "Codes for combined symbol-and-digit error control" *IEEE Tr. on Inf. Theory*, Vol.34, Nr.5, September 1988.

[Bracha 87-1] G.Bracha, "Asynchronous Byzantine Agreement Protocols" *Inform. and Comp.* Vol.75, Nov.1987, pp.130-143.

[Bracha 87-2] G.Bracha, "An $O(\log n)$ Expected Rounds Randomized Byzantine Generals Protocol" *Journal of the ACM* Vol.34, 1987, pp.910-920.

[Chen 78] L.Chen, A.Avizienis, "N-version Programming: A fault-tolerance approach to reliability of software operation" *8-th Annual Symposium on Fault-Tolerant Computing Systems*,pp3-9, Toulouse, France, June, 1978.

[Cristian 85] F.Cristian, H.Aghili, R.Strong, D.Dolev, "Atomic Broadcast: From Simple Diffusion to Byzantine Agreement", *15th INt, Conf. on Fault-Tolerant Computing*, 1985.

[Cristian 86] F.Cristian, H.Aghili, R.Strong, "Approximate clock synchronization despite ommission and performance faults and processor joins" *16th INt, Conf. on Fault-Tolerant Computing*, 1986.

[Davies 78] D.Davies, J.Wakerly, "Synchronization and matching in redundant systems",*IEEE Tr. on Comp. C-27,6* pp.531-539, June 1978.

[Dolev 81] D.Dolev,"Unanimity in an unknown and unreliable environment",*Proc. 22nd Annual Symposium on Foundations of Computer science* pp.159-168, 1981.

[Dolev 82-1] D.Dolev,"The Byzantine Generals strike again",*Journal of algorithms*,Vol.3, pp.14-30.

[Dolev 82-2] D.Dolev and H.R.Strong,"Polinomial Algorithms for multiple processor agreement", *Proc. 14th ACM SIGACT Symposium on Theory of Computing*, May 1982.

[Dolev 82-3] D.Dolev, M.J.Fischer, R.Fowler, N.A.Lynch and H.R.Strong,
"An Efficient Algorithm for Byzantine Agreement without Authentation" *Information and Control* Vol.52, No.2, pp257-274, 1982.

[Dolev 83-1] D.Dolev and H.R.Strong, "Autenticated algorithms for Byzantine agreement" *SIAM COMP*, Vol.12, No.4, pp.656-666, Nov. 1983.

[Dolev 83-2] D.Dolev, C.Dwork and L.Stockmeyer, "On the minimum synchronism needed for distributed consensus" *Proc. 24th Symp. Foundations of Computer Science* Tucson, Arizona, Nov. 1983,

[Fischer 82] M.Fischer and N.Lynch, "A lower bound for the time to assure interactive consistency" *Inform. Proc. Letters,* Vol.14, pp.183-186, 1982.

[Fischer 83] M.Fischer, N.Lynch and M.Paterson, "Impossibility of distributed consensus with one faulty processor" *Proc. 2-nd ACM Symp. on principles of database systems* 1983.

[Fischer 85] M.Fischer, N.Lynch and M.Paterson, "Impossibility of distributed consensus with one faulty processor" *Journ. of the ACM* Vol.32, No.2, pp.374-382, 1985.

[Fischer 86] M.Fischer, N.Lynch and M.Merritt. "Easy impossibility proofs for distributed consensus problems" *Proc 4th ACM Symp. Principles of Distributed Computing* Minaki, Canada, Aug. 1985, pp.59-70. (Journal of the ACM July 88)???

[Fischer 86] M.Fischer, N.Lynch and M.Merritt. "Easy impossibility proofs for distributed consensus problems" *Proc 4th ACM Symp. Principles of Distributed Computing* Minaki, Canada, Aug. 1985, pp.59-70. (Journal of the ACM July 88)???

[FTCS 71-90] "Digests of the Annual Symposium on Fault-Tolerant Computing" FTCS 1-18, 1971-1988, IEEE Computer Society.

[Gallager] L.E.Gallager, W.N.Toy, "The Fault-tolerant 3B-20 Processor", *National Computer Conference* pp.41-48,1981.

[Gils 85] W.J.van Gils, "How to cope with faulty processors in a completely connected network of communicating processors" *Information Processing Letters* Vol.20, Nr.4, May 1985, pp.207-213.

[Gils 86] W.J. van Gils, "A Triple Modular Redundancy Technique Providing Multiple Bit Error Protection Without Using Extra Redundancy", *IEEE Tr. on Comp.* Vol.C-35, Nr7, pp623-631,July 1986.

[Gils 87] W.J.van Gils, J.P.Boly, "On combined symbol and bit error control [4,2] codes over $\{0,1\}^8$ to be used in the (4,2) concept fault-tolerant computer." *IEEE Tr. on Inf. Theory* Vol. IT-33, Nr 6, November 1987.

[Gils 88] W.J.van Gils, "Design of error control coding schemes for three problems of noisy information transmission, storage and processing" Ph.D. thesis, Eindhoven University of Technology, Januari 1988.

[Hadz 87] V.Hadzilacos, "Connectivety requirements for Byzantine agreement under restricted types of failures" *Distributed Computing* Vol.2, 1987, pp.95-103.

[Harari] F.Harari, "Graph theory" *Addison-Wesley, Reading, Massachusetts,* 1972, ISBN 0-201-02787-9.

[Hill] F.J.Hill and G.R.Peterson, "Switching Theory and Logical design" John Wiley, New York, 1974.

[Hopkins] A.L.Hopkins, T.B.Smith, J.H.Lala, "FTMP-A Highly Reliable Multiprocessor for Aircraft", *Proceedings of the IEEE,* Vol.66,No.10,pp.1221-1239,Oct.1978.

[Knight 86] J.C.Knight, N.G.Leveson, "An emperical study of failure probabilities in multi-version software" *16-th Annual Symposium on Fault-Tolerant Computing Systems,*pp156-170, Vienna, July, 1986.

[Krol 82] "The (4,2) concept fault-tolerant computer" *12-th Annual Symposium on Fault-Tolerant Computing*, pp49-54, Santa Monica, CA, June, 1982.

[Krol 83] Th.Krol, "The (4,2) concept fault-tolerant computer" *Philips Tech.Rev.* No.41,pp1-11,1983.

[Krol 85] Th.Krol, W.J.van Gils, "The Input/Output architecture of the (4,2) concept fault-tolerant computer" *15-th Annual Symposium on Fault-Tolerant Computing*, pp254-259, Ann Arbor, MI, June, 1982.

[Krol 86] Th.Krol, "(N,K) Concept Fault tolerance" *IEEE Tr. on Comp.*, Vol.C35, No 4, April 1986, pp339-349.

[Lamp 82] L.Lamport, R.Shostak, and M.Pease,"The Byzantine General's Problem",*ACM Trans. Program. Lang. Syst.*, Vol.4, No.3, pp382-401, 1982.

[Laprie 82] J.C.Laprie, A.Costes, "Dependability: A Unifying Concept For Reliable Computing" *12-th Annual Symposium on Fault Tolerant Computing*, pp18-21, Santa Monica, CA, June, 1982.

[Laprie 85] J.C.Laprie, "Dependable Computing and Fault-tolerance: Concepts and Terminology", *Proc. 15th. Int. Symp. on Fault-tolerant Computing*, Ann Arbor, June 1985, pp.2-11.

[Lynch 82] N.Lynch, M.Fischer and M.Fowler, "A simple and efficient Byzantine Generals Algorithm" *Proc Second IEEE Symp. Reliability in Distributed Software and Data Base Systems* Pittsburg, Pennsylvania, pp46-52.

[Lund. 84] J.Lundelius and N.Lynch, "A new fault-tolerant algorithm for clock synchronization" *Proc Third Annual ACM Symp. Principles of Distributed Computing*, Vancouver Canada, Aug. 1984, pp.75-88.

[MacW 78] F.J.MacWilliams and N.J.A.Sloane, "The Theory of Error-Correcting Codes." Amsterdam, The Netherlands: North Holland, 1978.

[Melham 87] T.F.Melham, "Abstraction Mechanisms for Harware Verification" *Technical Report 106* University of Cambridge UK, Computer Laboratory, May 1987.

[Merritt 84] M.Merritt, "Elections in the presence of faults" *Proc Third Annual ACM Symp. Principles of Distributed Computing*, Vancouver Canada, Aug. 1984, pp.89-102.

[Pease 80] M.Pease, R.Shostak, and L.Lamport, "Reaching agreement in the presence of faults" *J. Assoc. Comput. Mach.*, Vol.27, No.2, pp228, 1980.

[Perry 85] K.J.Perry, "Randomized Byzantine Agreement" *IEEE Tr. on Softw. Eng.* Vol.11, No.6, pp.539-546, Jun 1985.

[Rabin 83] M.O.Rabin, "Randomized Byzantine Generals" *Proc. 24th Symp. Foundations of Computer Science* Tucson, Arizona, Nov. 1982, pp403-409.

[Shin 87] K.Shin and P.Ramanatan "Clock synchronization of a large multiprocessor system" *IEEE Tr. on Comp.* Vol.36, No.1, pp2-13, 1987.

[Siew. 78] D.P.Siewiorek et al., "C.vmp: A Voted Multiprocessor" *Proceedings of the IEEE*, Vol.66,No.10,pp.1190-1198,Oct.1978.

[Siew. 82] D.P.Sieviorek, R.S.Swarz, "The theory and practice of reliable system design" *Digital Press 1982*, Digital Equipment Corporation, USA.

[Sopho] "Special Issue on SOPHOMATION: The total approach to information management", *Philips Telecommunication Review*, Vol.43, No.2, June 1985.

[Srikanth 87] T.K.Srikanth and S.Toueg, "Optimal clock synchronization" *Proc 4th Symp. Principles of Distributed Computing* Minaki, Canada, Aug. 1985, pp.89-102. (Journal of the ACM July 88)???

[Toueg 87] S.Toueg, K.Perry and T.K.Srikanth, "Fast distributed agreement" *Proc 4th Symp. Principles of Distributed Computing*

Minaki, Canada, Aug. 1985, (SIAM J Computing, Vol.16 No.3 June 1987.)

[Wensley 78] J.H.Wensley *et al.*, "SIFT: design and analysis of a fault-tolerant computer for aircraft control " *Proc. IEEE*, Vol. 66, pp.1240-1255, Oct. 1978.

[patent] Th.Krol, (1978) USA patent 4,335,458 "Local error detection and aggregated correction".

[patent] Th.Krol, (1979) USA patent 4,402,045 "Fault-tolerant (4,2) computer requires less memory".

[patent] Th.Krol, B.J.Vonk, (1981) USA patent 4,512,020 "Improved code for the (4,2) concept, code 81".

[patent] Th.Krol, (1982) USA patent 4,633,472 "Error protection of the I/O memory in the (4,2) concept"

[patent] Th.Krol, W.J. v. Gils, (1984) Ned. patent 8402472 pending Can. patent 1,241,758 "Multiplicating incoming message in (N,K) concept".

Curriculum vitae

Thijs Krol was born on November 17, 1942, in Leeuwarden, The Netherlands. He studied electrical engineering at Eindhoven University of Technology, The Netherlands, from which he graduated cum laude on January 14, 1971. In 1971 he joined Philips Research Laboratories. From 1971 to 1985 he has been working in different areas of fault-tolerant computing, error-correcting codes and telecommunication switching systems. Since 1985 he is mainly involved with specification and high-level hardware description languages for VLSI design.
From 1983 to 1990 he has been guest-lecturer in fault-tolerant computing at the Eindhoven University of Technology.

Stellingen

behorende bij het proefschrift

A Generalization of Fault-Tolerance Based on Masking

van

Thijs Krol

Eindhoven
24 september 1991

1. In tegenstelling tot fouten tolererende systemen die gebaseerd zijn op ver-N-voudiging, biedt een fouten tolererend systeem dat gebaseerd is op het (N, K) concept, de mogelijkheid om de totale hoeveelheid hardware of de call-rate van het systeem te minimaliseren door de verhouding tussen de hoeveelheid hardware voor het geheugen en de hoeveelheid hardware voor de processoren te varieren.
   Het (N, K) concept biedt tevens de mogelijkheid het aantal benodigde modules voor fouten tolerantie aan te passen aan het aantal modules dat nodig is voor consistente communicatie met de omgeving.

   [dit proefschrift]

2. De toepassing van het (N, K) concept wordt beperkt door de tijd die nodig is voor decoderen.

   [dit proefschrift]

3. Wanneer geen bebruik gemaakt wordt van authenticatie is het Input Probleem en het Byzantijnse Generaals Probeem in praktijk tot nu toe alleen oplosbaar wanneer het aantal te tolereren fouten minder is dan vier.

   [dit proefschrift]

4. Van alle tot nu toe bekende synchrone deterministische Byzantijnse Generaals Algorithmen, die geen gebruik maken van authentiseren, belasten de algorithmen, die gebaseerd zijn op fouten corrigerende codes, het communicatienetwerk tussen de modules het minst.

   [dit proefschrift]

5. Uitspraken over betrouwbaarheidsverbeteringsfactoren hoger dan 10^3, die uitsluitend gebaseerd zijn op berekeningen zonder experimentele verificatie, moeten in het algemeen als fabels worden beschouwd.

6. Een fout-hypothese is een uitstekend middel om minder gunstige praktijk resultaten te verbergen.

7. Het woord fout in "fouten tolererende digitale systemen" suggereert ten onrechte dat dergelijke systemen ook bestand zouden zijn tegen foute input gegevens.
   Het is daarom beter te spreken van "systemen die in staat zijn incorrect gedrag van de componenten waaruit ze zijn opgebouwd te tolereren".

8. De voorliefde van wiskundigen om hun bewijzen zo elegant mogelijk op te schrijven, maakt deze bewijzen voor hen die aan de zijlijn van de wiskunde werken vaak moeilijk toegankelijk.

9. De hoeveelheid wetenschappelijke publicaties op het gebied van zelfcontrolerende logica is meer bepaald door de geschiktheid van het onderwerp voor doctoraalscripties en proefschriften, dan door het practisch belang van het onderwerp.

10. De langzame invoering van het gebruik van formele methoden voor het ontwerpen van digitale systemen wordt mede veroorzaakt door de beperkte kennis op het gebied van wiskunde en informatica bij de ontwerpers en door gebrek aan kennis van het ontwerpproces bij de informatici en wiskundigen.

    [Proceedings IFIP WG 10.2-WG10.5 International Workshop on Applied Formal Methods For Correct VLSI Design, Houthalen, Belgie, 13-16 November 1989]

11. Een project georganiseerde research organisatie kan slechts voorspelbare resultaten opleveren. Mits aan de juiste omgevingsvoorwaarden is voldaan heeft daarentegen een discipline georganiseerde research organisatie de potentie in zich tot werkelijke vernieuwing.

12. Wetenschappelijke vooruitgang wordt vaak verkregen door het poneren van stellingen en daarna een ander het tegendeel te laten bewijzen.