# Linear Protection Switching Requirements Simulation & Verification

## Graduation report

**J.L.R. de Graaff**
29 August, 1997

Delft University of Technology
Faculty of Electrical Engineering
Telecommunications and Traffic-Control Systems Laboratory

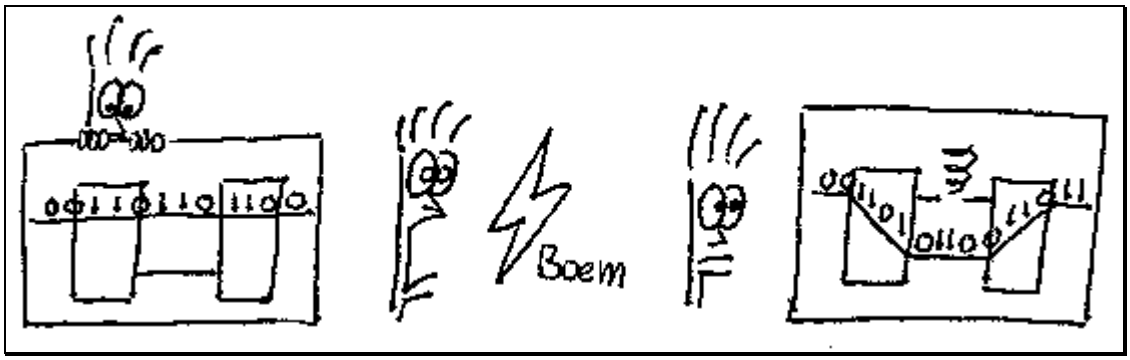
**Supervisor** ir. J.A.M. Nijhof
**Graduation no.** A-786

Lucent Technologies
PRC Platform
Huizen, The Netherlands

**Supervisor** ir. F. Speelman
**Technical mentor** ir. M.P.J. Vissers
**Coach** ir. S.L. Goede
**Document** JNL 241-R-97003

# Linear Protection Switching Requirements Simulation & Verification


Graduation report
Author: J.L.R. de Graaff
College number: 275398
Report date: 26 February, 2001
Graduation date: 29 August, 1997




A sketch of protection switching, as used on the graduation presentation invitation card for friends and family.

# Acknowledgments

This report describes my graduation research at Lucent Technologies. The research is done in the Cross Product System Engineering group of the Platform organization.

The graduation research is supported from the Telecommunications and Traffic-Control laboratory of the faculty of Electrical Engineering of the Delft University of Technology.

System specification modelling and verification is relatively young area and there is a great need for methods and tools. Additionally, there was need for the verification of the linear protection switching protocol described in ETSI ETS300-417 specification. My graduation project is about both subjects.

Besides reporting about research results, this report intends to give an overview of
- Transport networking (SDH in particular)
- Linear protection switching
- The APS protocol
- The verification tool SPIN and its language PROMELA.

I would like to thank Jos Nijhof for his willingness to supervise my graduation project, Frans Speelman for his open-minded and pleasant guidance, Maarten Vissers for his enthusiastic support, my room-mate Wouter Boeke for his constructive criticism and interesting lunch breaks and last but not least my coach Sam Goede for the good conversations and for his guidance.

Also I would like to thank Maarten Vissers for the cake he treated when I found the fifth error in the specifications. This little (and pleasant) contest was placed to encourage me to find as much errors as possible.

Finally I would like to thank my friends Berend Vosmer, Erwin Wils and especially Patricia Lansbergen for their support and discussions in the final period of my study.

In annex E, a paper is included in a standard publication style. It is part of the graduation procedure to deliver such a publication covering the graduation work. Of course, this can also be used as an abstract for this report.

# Contents

# Abstract

The electronic systems we design today, are becoming larger and more complex. This is especially true for telecommunication systems. Such systems are first designed at a functional level. This is described in specifications. These system specifications are used for further development of the system. It is of great importance that these specifications are correct. Errors in specifications could result in significant costs if they are detected in further development, and in even more problems if they are not.

Describing system behavior calls for modelling methods and specification languages. System behaviour has multiple aspects and each demand specific modelling techniques. Examples are functional behaviour, the behaviour in time and the architectural structure. To determine whether specifications are correct or not, they must be verified. System verification can principally be done by reasoning or by structural analysis, but for larger and more complex systems, this is not feasible any more. Automated verification methods are needed and the computer is used for this purpose.

System simulation gives great insight in its behaviour, but in order to determine the absolute correctness of system specifications, a more thorough method is needed. Verifying that all possible states of the system are correct, is called validating. In order to validate a system, correctness requirements must be specified. These requirements are used by the verification tool to analyze the system in an exhaustive manner to proof its correctness.

A problem with this method is called the state explosion problem. The verification tool translates the system description into a finite state machine, after which it can analyze its states. For large and complex systems, such translation can result in an enormous number of states. Limitations prevent such a full state search. Common limitations are the available memory (to store which state is analyzed) and the maximum time a validation may take. Another method must be introduced, called state reduction. The verification based on this technique is called proof approximation. The verification tool SPIN supports state reduction techniques and is capable to perform all mentioned verification techniques. Its modelling language, PROMELA, has all expression power that is needed for modelling distributed systems.
With this tool the linear protection switching specifications are verified.

Protection switching is a method to provide a certain level of survivability in a transport network by means of diversity. When a transmission line fails, its traffic is switched over a pre-assigned redundant transmission line, called protection line. In linear protection switching two nodes coordinate this protection switching, by means of a protocol. The specification of this protocol is verified. It has resulted in the detection of several errors and new insights.

Another issue in communication network simulation is the graphical presentation of the behaviour. A method is developed that enables the design of a graphical model and the presentation of simulation results in that model. The basis of this method is that the graphical model is defined independent of the system model. This results in the freedom to determine the abstraction level at which the simulation presentation is desired and the shielding of certain system behaviour details.

# Samenvatting

De elektronische systemen, die we tegenwoordig ontwerpen, worden groter en complexer. Dit geldt in het bijzonder voor telecommunicatiesystemen. Zulke systemen worden eerst ontworpen op een functioneel niveau. Dit wordt beschreven in specificaties. Deze systeemspecificaties zijn de basis voor de verdere ontwikkeling van het systeem. Het is bijzonder belangrijk, dat deze specificaties correct zijn. Fouten in specificaties kunnen in aanzienlijke kosten resulteren als ze worden gevonden in de verdere ontwikkeling, en zelfs in grotere problemen als ze niet worden gevonden.

Om systeemgedrag te kunnen beschrijven zijn er methoden en specificatietalen nodig. Systeemgedrag heeft meerdere aspecten and elk vraagt om specifieke modelleringstechnieken. Voorbeelden zijn functioneel gedrag, het gedrag in de tijd en de structuur van de architectuur. Om te kunnen bepalen of specificaties correct zijn, moeten ze worden geverifieerd. Systeemverificatie kan in principe door redeneren of structurele analyse gedaan worden, maar voor grotere en complexere systemen wordt dat ondoenlijk. Daar zijn automatische verificatiemethoden voor nodig en hiervoor wordt de computer gebruikt.

Systeemsimulaties geven een goed inzicht in het gedrag, maar er is een grondigere methode nodig om te kunnen bepalen of de systeemspecificaties absoluut correct zijn. Het verifiëren van alle mogelijke toestanden van een systeem, wordt valideren genoemd. Om een systeem te kunnen valideren zijn correctheidsregels nodig. Deze regels worden door het verificatieprogramma gebruikt om het systeem op een uitputtende manier te analyseren, zodat de correctheid bewezen kan worden.

Een probleem van deze methode is het toestand explosie probleem. Het verificatieprogramma vertaald de systeembeschrijving in een ‘finite state machine’, waarna de toestanden geanalyseerd kunnen worden. Voor grote en complexe systemen, kan dit resulteren in een enorme hoeveelheid toestanden. Beperkingen staan een volledige toestandsanalyse in de weg. Veel voorkomende beperkingen zijn de hoeveelheid geheugen (om de toestanden die geanalyseerd zijn op te slaan) en de maximum tijd die de validatie mag duren. Een andere methode, genaamd toestandsreductie, is nodig. De verificatie gebaseerd op deze techniek wordt bewijsbenadering genoemd. Het verificatieprogramma SPIN gebruikt deze toestandsreductie technieken en is in staat om al de genoemde verificatie technieken uit te voeren. Haar modelleringstaal, PROMELA, heeft al het benodigde expressievermogen om gedistribueerde systemen te modelleren.
Met dit programma wordt de ‘linear protection switching’ specificaties geverifieerd.

‘Protection switching’ is een methode om een bepaald beschikbaarheidsniveau te bereiken door middel van diversiteit in een transport netwerk. Als een transmissielijn faalt, wordt het verkeer daarvan geschakeld over een van te voren toegekende redundante transmissielijn, die ‘protection’ lijn wordt genoemd. Bij ‘linear protection switching’ coördineren twee knooppunten van het netwerk deze ‘protection switching’, door middel van een protocol. The specificatie van dit protocol wordt geverifieerd. Het heeft geresulteerd in het vinden van verscheidene fouten en het verkrijgen van nieuwe inzichten.

Een andere kwestie bij het simuleren van communicatienetwerken, is de grafische presentatie van het gedrag. Een methode is ontwikkeld, die het mogelijk maakt om een grafisch model te ontwerpen en de simulatieresultaten in dat model te presenteren. Uitgangspunt van deze methode is dat het grafisch model onafhankelijk van het systeemmodel gedefinieerd kan worden. Dit geeft de vrijheid om te bepalen op welk abstractieniveau de simulatiepresentatie is gewenst en om bepaalde details van het systeemgedrag te verbergen.

# List of Figures

# List of Tables

# 1 Introduction

The electronic systems we design today, are becoming more and more complex. This is especially true for telecommunication systems. Generally, the behaviour and functionality in the first stages of specification, will be described in the plain English language. In a later stage, when more detailed information is necessary, a more formal way of specification semantics will be required.
From the highest level of behavioral system description, we will derive a more detailed behavioral decomposition in order to come to a set of implementable system requirements. Furthermore, our systems have to be compatible with related telecom systems on network level.

A first and most important issue in specification is the specification itself. How can system behaviour be modelled? What are the characteristics of the aspects of system behaviour? What languages are needed? Can we model a system in one language or do we need different languages for different aspects of the system? And how do these languages interact?
If a system cannot be modelled adequately, problems are not far away.

The second issue is: How do we know if these specifications are correct? Do these specifications actually describe the intended behaviour? Errors in these specifications can be disastrous for the functioning of the system, either on a equipment level or on the network level. When for example an error in a protocol, on which a large public telephone network is relying, results in breaking down of all communications, thousands of users are cut off. Such catastrophes results in significant loss of revenues or even human lives.

Verification of system specifications is an absolute must. Besides the determination of correctness of specifications, the verification process also adds insight and confidence in the system.

The objectives of this research are:
1. an investigation how system specifications can be modelled and how they can be verified,
2. the verification of the linear protection switching specification from the ETSI ETS300-417 document, the interpretation of the findings and an investigation of a possible solution,
3. an investigation in improvements of the verification environment.

The report has the following structure:
In chapter 2 the system engineering process is described and system specification is investigated.
In chapter 3 transport networks and several related concepts are discussed.
In chapter 4 is protection switching discussed.
In chapter 5 the investigation in system verification methods is presented.
In chapter 6 the results of the verification of the ETSI specifications are discussed.
In chapter 7 is graphical simulation as an improvement for the verification process is discussed.

# 2 System Specifications

Describing a system is a complex task that has many pitfalls. In this chapter the character of system specifications is discussed. First the system engineering process is described (2.1) and then the properties of system specifications are examined (2.2).

## 2.1 System Engineering

In [Arn94] system engineering is defined as:

> *'the technical discipline concerned with selection and development of an optimum overall solution to a functional problem defined in a real environment'*

The system engineer is primarily concerned with the functional behaviour at a certain abstraction level. Implementation details are not considered, other than that limitations are considered and specific technology choices are accepted.

In [NW96] a system realization structure was presented, which is repeated Figure 2-1. Within Lucent there is often referred to this structure. Particularly in designing telecommunication systems, system engineering has much to do with external standardization documents. The Tier-x notation refer to the documents that are passed on to the next design/development phase.

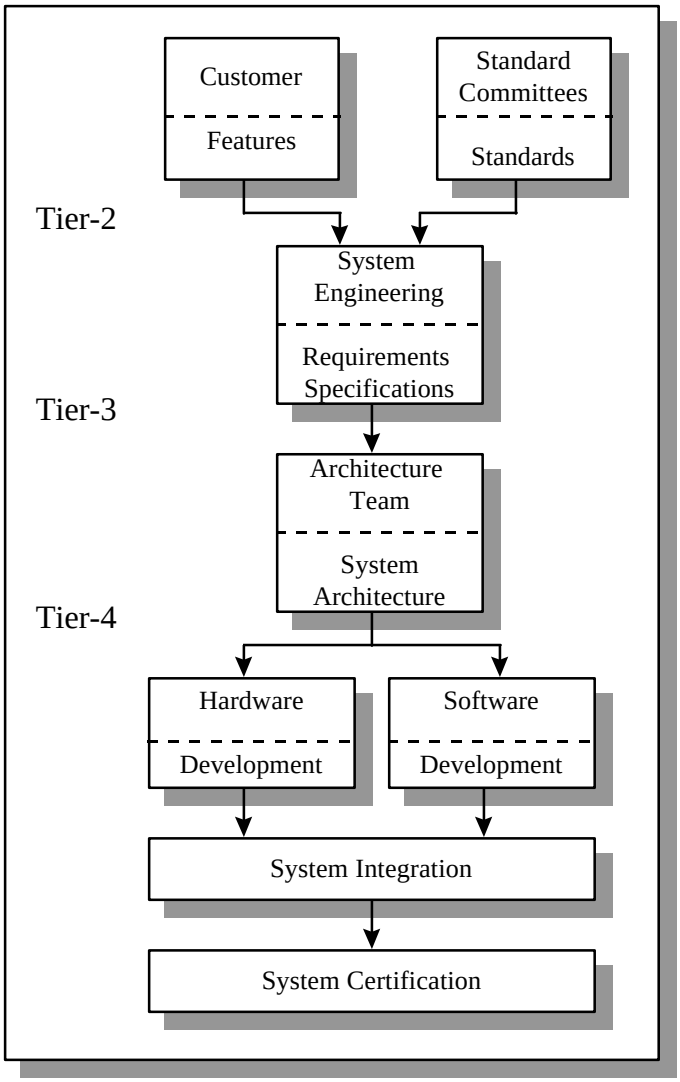



**Figure 2-1 System realization structure**

As systems become larger and more complex, the system engineering and architecture discipline is becoming more important and it is more difficult to verify. Especially public telecommunication systems, that inherently have a large number of users, demand an increasing effort to be designed.

## 2.2 System Specifications

The description of a system is called a system specification. It is crucial that these specifications are correct, because they are used as a starting-point for implementing the system. Design errors made in this design phase, commonly surface if the system is (partially) implemented. The later an error surfaces, the more difficult it is to correct or the more expensive it is to return to this phase.

Successful specifications are:
A) Complete, correct, unambiguous and consistent
B) Minimal
C) Usable
D) Identifiable, traceable and surveyable
E) Verifiable and testable for properties
F) Executable
G) Easily modified
H) Linkable to other requirements

**Figure 2-2 Requirements of successful specifications**

In Figure 2-2 a list of requirements is presented, that are desirable for specifications. Some of these requirements conflict and that makes it hard for to satisfy all of them.

Requirement A is the fundamental purpose of specifications. Requirement B addresses over-specification. Requirement C addresses implementability of the specifications. Requirement D demands that specifications can be understood and related to common terms and concepts. Requirement E demands the specification to be testable; this requires that the specification also formulates *what exactly* is correct. Requirement F demands the specification to be understandable by the computer. Requirement G and H address the maintainability and transportability of specifications.

The worst verification is verification by reasoning. The correctness is obvious because the designer has designed the system to meet the requirements he has in his mind. Exactly what he has not in mind, is the common cause for errors.

It is essential that the specifications are entered as formal as possible. Formal specification demands the designer to look at the system in a precise and complete way. When the system is specified formally, it can easily be verified with use of the computer. Another property of formal verification is that besides the specification of the system a specification of correct behaviour is necessary. To view the system from a viewpoint of absolute requirements rules, also adds insight.

# 3 Transport Networks

In this chapter an overview of transport networking is presented (3.1). Several concepts that are needed to discuss its characteristcs are presented (3.2) and specifically the Synchronous Digital Hierarchy is discussed (3.3), which we need as a basis for the examination of protection switching.

## 3.1 Introduction in Transport Networks

A *transport network* is a network system that is concerned with the transport of information from one location to another. Connections are set up by a network manager. Connections are of fixed bandwidth, have a fixed route and exist until broken down by the network manager.

A common purpose of a transport network is the transportation of telephone traffic. When for example subscriber A wants to call subscriber B, then the exchange of subscriber A switches the call over a connection in the transport network to the exchange of subscriber B. This is illustrated in Figure 3-1.
Note that the transport network does not switch the actual traffic, it only provides fixed connections. One could call this *static routing*. This is a fundamental difference with data networks. Data networks route all traffic *dynamically* and can be much more efficient, because the traffic is allocated the exact needed bandwidth and unused bandwidth is available for other traffic.

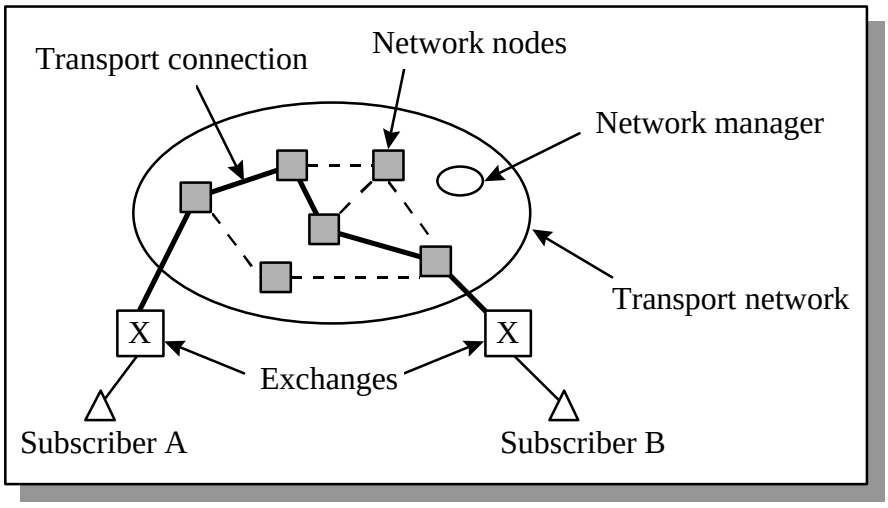



**Figure 3-1 Information transport in a transport network**

Managing large transport networks could become very complex. Two concepts to manage the complexity are:

- Layering
- Partitioning.

A transport network can be decomposed into a number of independent transport layer networks with a client/server association between adjacent layers. Each layer network can be separately partitioned in a way which reflects the internal structure of that layer network or the way that it will be managed. Thus the concepts of partitioning and layering are orthogonal, as shown in Figure 3-2.

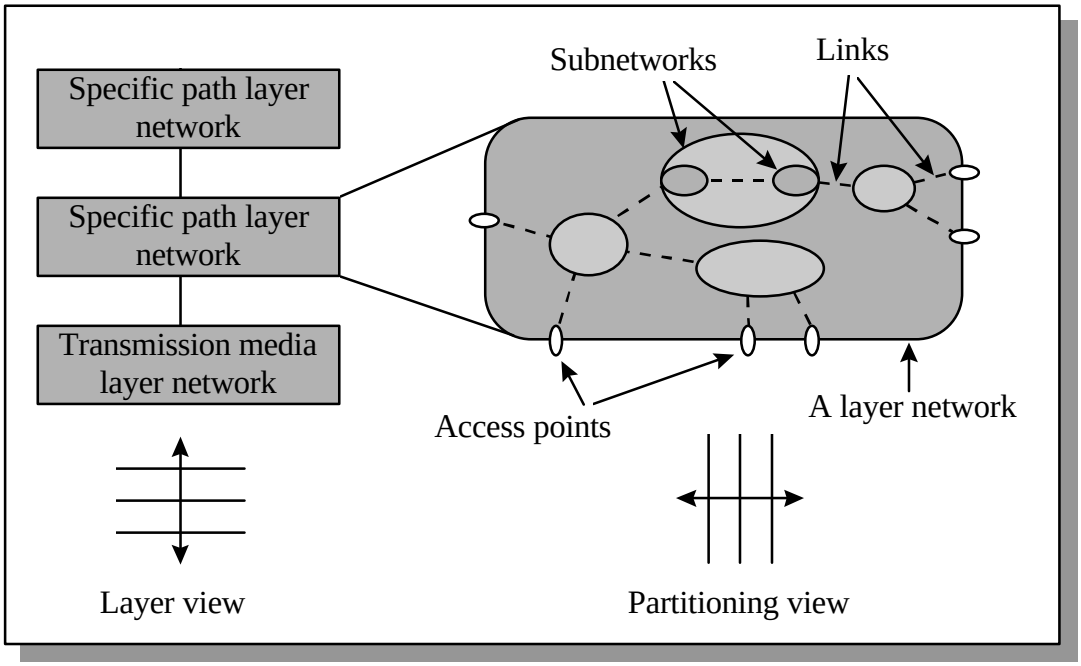



**Figure 3-2 The layering and partitioning concept.**

**Layering**

Layering allows the transport network functionality to be described hierarchically as successive levels.

The layering concept of the transport network allows:
- each layer network to be described using similar functions,
- the independent design and operation of each layer network,
- each layer network to have its own operations, diagnostics and automatic failure recovery capability,
- the possibility of adding or modifying a layer network without affecting other layer networks from the architectural viewpoint,
- simple modelling of networks that contain multiple transport techniques.

The ITU[1] uses the layer categories in Figure 3-3 to divide networks into specific layers.

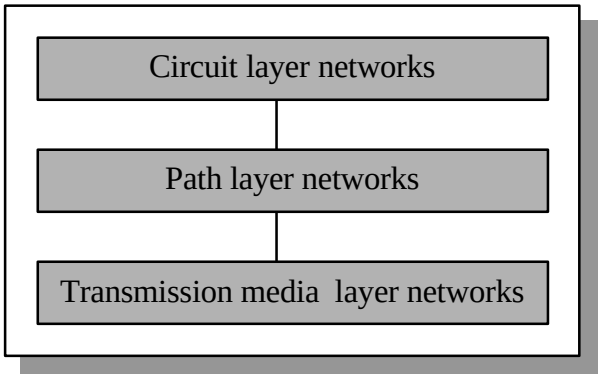



**Figure 3-3 The layer categories of a layered network**

The lowest layer network category is the transmission media layer. It's connections consist of the physical transportation facilities such as: fiber links, electrical links (coaxial cable), radiowave links or satellite links. These layers are primarily concerned with modulation, synchronization and regeneration.

The path layer networks primarily provide the flexibility of the network. Paths can be set up and torn down as they circuit layer networks need them.

The highest layer network category is the circuit layer. This layer network provides users with direct telecommunications services such as circuit switched and leased line services.

Note that layering should *not* be confused with the layers of the OSI model. The layers described here each form an independent network, while the OSI model layers represent services at different abstraction levels in *one* network.

**Partitioning**

The partitioning concept enables a network layer to be divided into subnetworks. The partitioning concept is recursive; subnetworks can be partitioned in smaller subnetworks interconnected by links.

The partitioning concept is important as a framework for defining:
- The network structure within a layer.
- Administrative boundaries between network operators jointly providing connections within a single layer network.
- Domain boundaries within a layer network of a single operator to allow the apportioning of performance objectives to the architectural components.
- Routing domain boundaries within the layer network of a single operator.
- The part of a layer network or subnetwork that is under control of a third party for routing purposes (e.g. customer network management).

[1] International Telecommunication Union

Examples of transport networks are:
- the Synchronous Digital Hierarchy (SDH),
- the Synchronous Optical Network (SONET),
- the Asynchronous Transport Mode (ATM) network..

SONET is an ANSI[1] standard. SDH is an ETSI[2] standard and developed to be compatible with SONET. Therefore there is a great resemblance between SDH and SONET. An important difference is that SDH is accepted as a world-wide standard.
ATM is a relative new transport technique and is fundamentally different from SDH and SONET, in that the communications are asynchronous. It is particularly efficient in transporting traffic with different bandwidths and characteristics. Although the actual transport is different, a lot of concepts from the synchronous world also apply to ATM.

## 3.2 Functional Modelling

Functional modelling is a methodology to decompose network functionality into a collection of functions. There are three types of functions, called atomic functions, of which a network (layer) can be described. Each represents a characteristic operation. Functional modelling is also very useful in describing the layering and partitioning concepts.

The atomic functions are:
- the connection function,
- the adaptation function,
- the termination function.

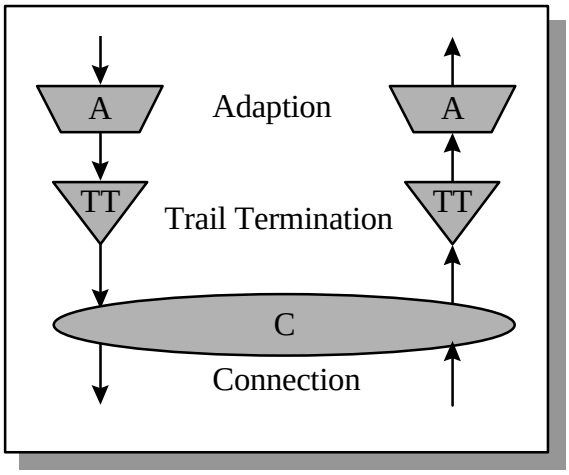



**Figure 3-4 The atomic functions and their interconnection**

The atomic function interconnection is always as shown in Figure 3-4. The direction from trail termination function towards connection function is called *source direction*. The direction from connection towards trail termination function is called *sink direction*. This is illustrated in Figure 3-5. Often, the unidirectional links are grouped and presented as bidirectional links.

[1] American National Standards Institute
[2] European Telecommunications Standards Institute

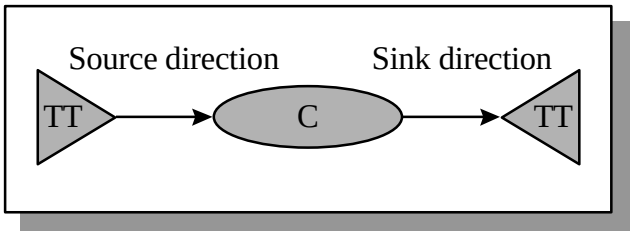



**Figure 3-5 Source and sink directions**

In Figure 3-6 a network fragment is shown to illustrate functional modelling. Each layer is described by a set of atomic functions. Note that the adaptation function is actually an inter-layer function, presenting the conversion between a client and server layer. It is usually drawn in the server layer.

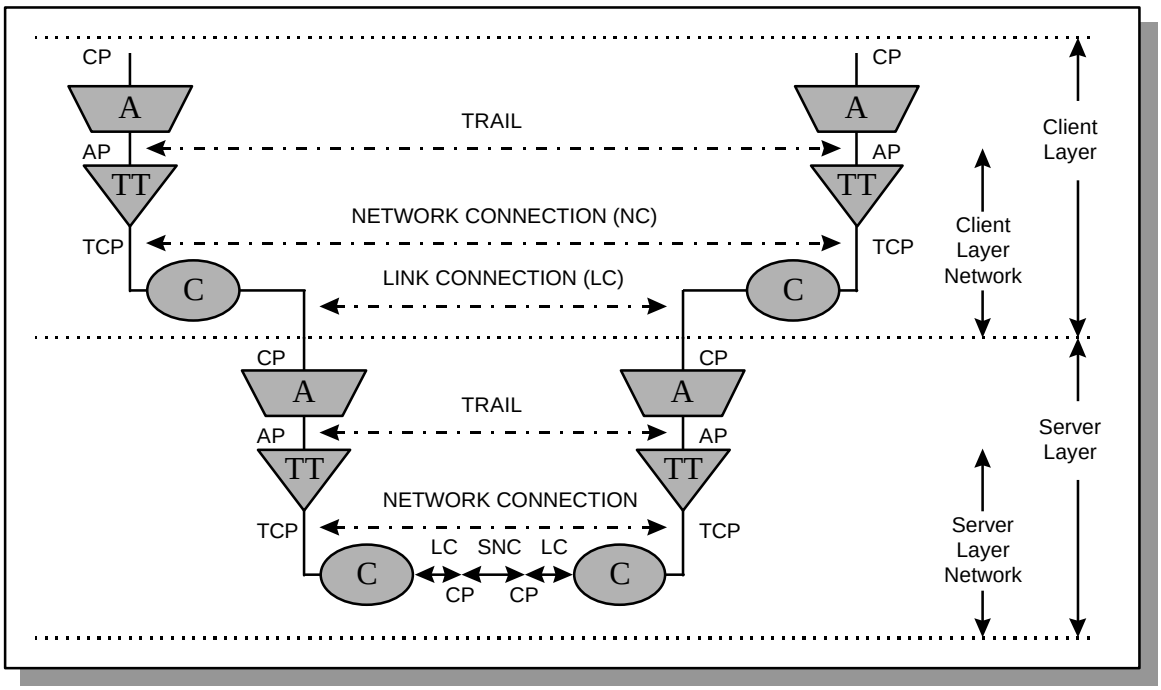



**Figure 3-6 Network fragment illustrating functional modelling**

The fundamental entity in a layer network is the trail. It is a transport entity responsible for the integrity of transfer of characteristic information. Characteristic information in a network (layer) is a signal of a specific rate and format.

A trial is called a *circuit* in a circuit layer network, a *path* in a path layer network and a *section* in the section layer network. Section layer networks are part of the transmission media layer network category (this is discussed in the next paragraph).

Each atomic function is discussed and its characteristic operation is explained.

**Connection function**

Provides flexibility within the layer. It may be used by the network operator to provide:

| | |
|---|---|
| *routing* | determination and setting up connections. |
| *grooming* | the allocation of server layer trails to client layer connections which groups together client layer connections whose characteristics are similar or related. |
| *protection* | explained in paragraph 4.1 |
| *restoration* | explained in paragraph 4.1 |

**Trail termination function**

Performs the signal integrity supervision of the layer.

In the source direction it generates and adds:
- *error detection code*
- *trail trace identifier*

In the sink direction, it monitors:
- *bit errors*
- *(mis)connections*
- *near- and far-end performance*
- *server signal fail*
- *signal loss*

**Adaptation function**

Represents the conversion process between a server and a client layer. Processes that may be present are:

| | |
|---|---|
| *scrambling/descrambling* | alters the digital signal to ensure a sufficient density of 0→1 and 1→0 transitions to allow bit clock recovery from it. |
| *encoding/decoding* | adapts a digital signal to the characteristics of the physical channel. |
| *alignment (frame, pointer interpretation)* | locates the start of a frame or the pointer location and reports a loss of frame or pointer (LOF, LOP) |
| *bit rate adaptation* | accepts a different bitrate than the output bitrate by means of creating or removing gaps. |
| *frequency justification* | accepts an input at a different frequency (and/or phase) than the output by writing data into allocated justification bits. |
| *multiplexing/demultiplexing* | information of multiple sources end up in pre-allocated time slots in the resulting time-division signal. |
| *timing recovery* | extracts a clock signal from an incoming signal. |
| *smoothing* | filters the phase step of ‘gapped input signals’. |
| *payload identification* | the insertion and checking of Trail Signal Label (TSL) to allow different client signals. |

*Reference points* are defined as delimiters of the function blocks.
They are:

| | |
|---|---|
| **Access Point (AP)** | between adaptation and termination function |
| **Connection Point (CP)** | between connection and adaptation function |
| **Termination Connection Point (TCP)** | between termination and connection function |

Between the reference points, information is transported over *transport entities*. Two basic entities are distinguished according to whether the information is monitored for integrity. These are trails and connections. Connections can be further distinguished in three types.

The different transport entities are:

| | |
|---|---|
| **Trail** | represents the transfer of monitored adapted information between access points. |
| **Network connection** | is capable of transferring information across a layer network. It is delimited by TCPs. It is formed from subnetwork connections and/or link connections. |
| **Subnetwork connection** | is capable of transferring information across a subnetwork. It is delimited by CPs at the boundary of the subnetwork. |
| **Link connection** | is capable of transferring information across a link. It is delimited by ports (CPs) and represents a fixed relation between the ends of a link. A link connection represents a pair of adaptation functions and a trail in the server layer network. |

The largest subnetwork connection on a trail is a network connection, the smallest a link connection.

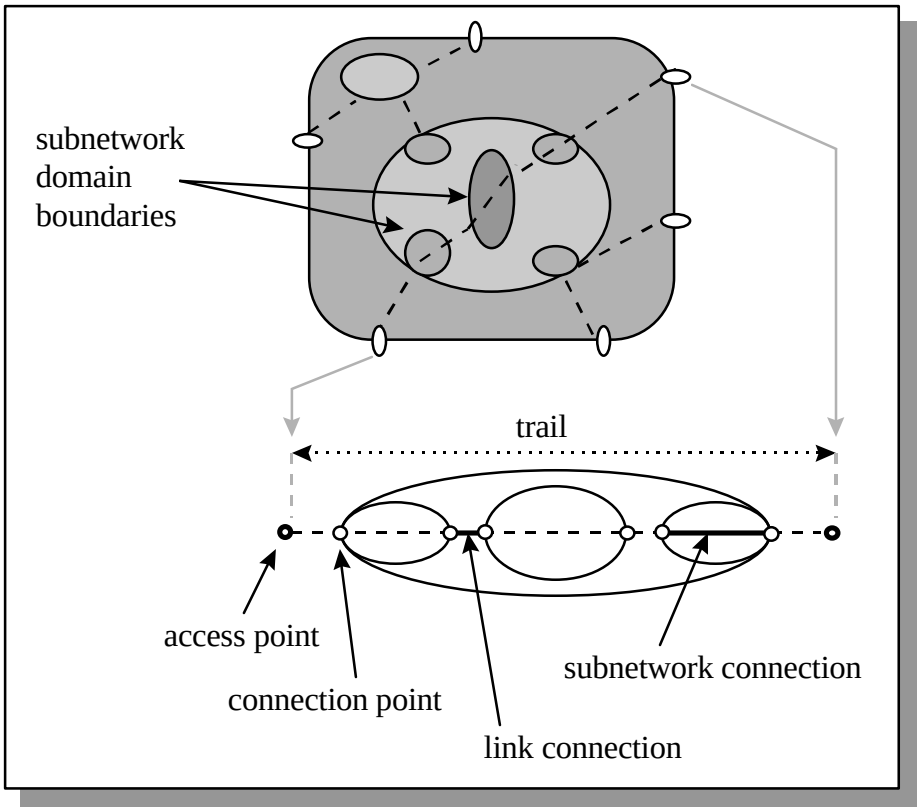



**Figure 3-7 The trail and connections of a network layer**

## 3.3 Synchronous Digital Hierarchy (SDH)

SDH is developed with the following objectives:

- become a world-wide synchronous digital transport network standard.
- capable of transporting the (at that time) widespread PDH[1] signals efficiently.
- Full connectivity on all levels such that network operators are free in choosing equipment from any manufacturer.
- Enhanced Operation, Administration and Maintenance (OA&M) capabilities by the allocation of overhead capacity in the SDH frame.

In order to be capable of transporting the different regional PDH bitrates, SDH has a complex multiplexing structure. This structure goes hand in hand with the network layering. See [Hol91] for more information.

The SDH network layers are shown in Figure 3-8:

- Lower order path layer (LP)
- Higher order path layer (HP)
- Multiplex section layer (MS)
- Regeneration section layer (RS)
- Optical/electrical section layer (OS/ES)

[1] Plesiochronous Digital Hierarchy; a near synchronous communication: accommodates traffic sources with (slightly) different clocks and different path delays by means of justification.

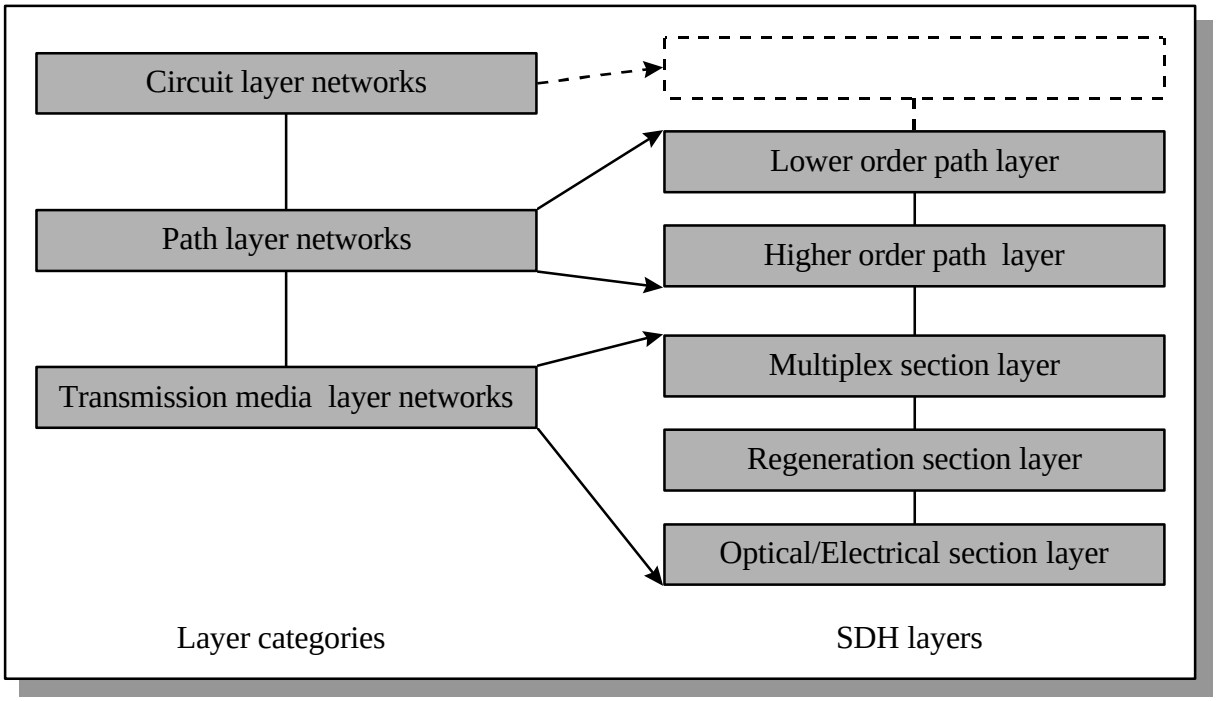



**Figure 3-8 The SDH network layers**

The Multiplex section layer, together with the layers beneath: the Regenerator section layer (signal regeneration between communication links) and the Physical media Layer (fiber, coaxial cable or radio link), form the transmission media layer.

SDH has two path layers to achieve the necessary flexibility in multiplexing the different PDH signals.

A limited overview of the SDH frame structure is shown in Figure 3-9. It has no intention to be a complete description, it only explains the basic terms. For a more complete overview the reader is referred to [SR91].

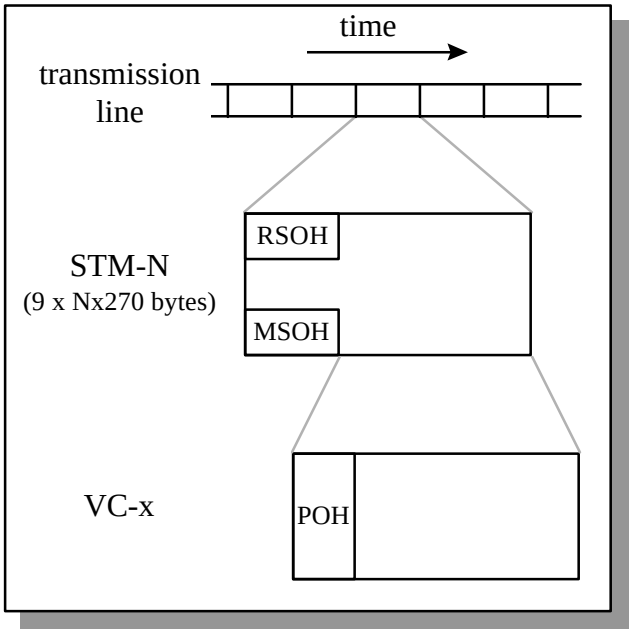



**Figure 3-9 SDH frame structure**

In the MS layer frames of 125 µs duration are transported. This is an inheritance from the PDH networks. Such a frame as called Synchronous Transfer Module (STM). There are several levels of modules, denoted by STM-N with N=1, 4, 16 and 64, corresponding to different signaling rates. As in Figure 3-9, the frame is usually drawn in matrix form. The matrix presents rows and columns of bytes. The frame is transmitted on a per row basis.

In the STM frame an overhead defined, called section overhead (SOH). This overhead consists of two parts: the regeneration section overhead (RSOH) and the multiplex section overhead (MSOH).
In the STM frame smaller structures are transported, called Virtual containers (VC). These VCs form the communication in the path layers. In the VC is also overhead defined: path overhead (POH).

To satisfy the requirements for the transfer of OA&M messages across the network nodes, a Data Communications Channel (DCC) is defined within the multiplex section overhead. Based in these data channels, a protocol stack, according to the ISO OSI model, is defined, called Embedded Communications Channel (ECC). Network management systems use this packet network to observe and control the network.

# 4 Protection Switching

With transport networks several methods are used to guarantee the availability of transport services. In this chapter such a method, called protection switching, is discussed. First an general introduction into protection switching is presented (4.1). Then the operation and modes of linear protection switching is discussed (4.2). In 4.3 the aspects of linear protection switching in SDH networks are discussed and in 4.4 the details of the APS protocol are explained. The ETSI linear protection switching specifications are discussed in 4.5.

## 4.1 Introduction in Protection Switching

**Failures and survivability**

Protection switching is a method to provide a higher level of survivability in a transport network by means of diversity. Network survivability is defined as the ability to provide continuity of transport services in the presence of failures.

Some examples of failures are:
- Hardware failures
- Accidents
- Architecture/protocol errors
- Maintenance errors
- Management procedural errors

The causes for failures can be split in three categories as is done in ITU recommendation M.495:
- Equipment failure (this can be reduced by improving equipment reliability).
- Outages due to operating organization. For example, maintenance work or human errors.
- External causes which are very difficult to prevent and for which specific protection might be needed.

It is assumed that design errors belong to the first category.

One quality measure of a network is availability. Availability is the percentage of total time that a service is available, taken over a long period of time.

**Availability enhancement techniques**

There are two ways to automatically recover without waiting on the problem to be located and fixed, namely *protection* and *restoration*. Protection is a method of alternate capacity selection that makes use of preplanned and reserved capacity in the network. In case of restoration the redundant capacity is not preassigned and must be ‘discovered’ by some network intelligence, which uses a method to select alternate capacity based on the current network status.

An important distinction is based on response time. Restoration of service after failure has usually been associated with a slow, often manual response, while protection is assumed to be fast and autonomous. Today, automatic protection switching response times are less than 50 milliseconds while restoration based on digital cross-connect systems can take many minutes.

**Object of protection**

In protection switching, *equipment* and *transmission* protection switching can be distinguished. Equipment protection is concerned with the selection of equipment parts. Equipment protection takes place if a system detects a hardware failure. Redundant parts take over from the failed parts within the system. Transmission protection switching is concerned with selection of received signals, based on quality thresholds.

Although the mechanisms for equipment and transmission protection switching are basically the same, only the latter is a topic in standardization. Equipment protection switching is left to the vendors, as an implementation aspect of high reliability and availability requirements per system. In many cases and also in this report the term protection switching is used instead of transmission protection switching.

**Protection switching process**

The protection switching operation is achieved by the protection switching process which makes use of one or more pre-assigned protection entities that can be used to replace one or more working entities in case of a failure by means of protection switches. In a transport network the working and protection entities are transport entities, for example a complete transmission line between two nodes or a single transmission channel passing many nodes and links.

**Protection topology and network layer topology**

A topology describes the logical interconnection patterns of ‘nodes’ with their interconnecting ‘links’. Some basic topologies are shown in Figure 4-1.

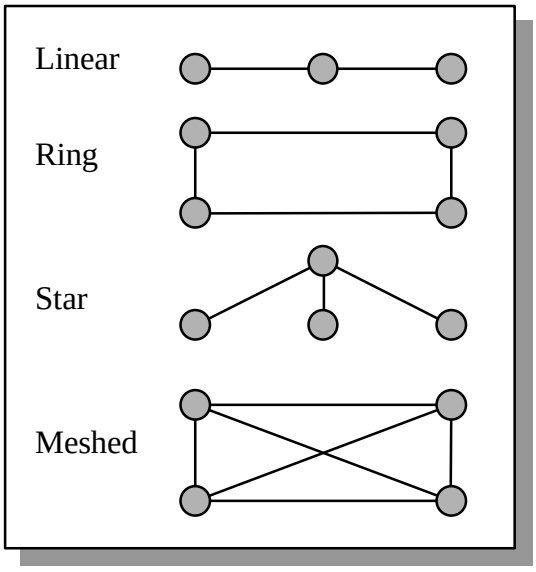



**Figure 4-1 Network topologies**

In case of protection topology the ‘nodes’ are the protection switches and the ‘links’ comprise both the ‘working’ and ‘protection’ transport entities. A working transport entity is pre-assigned to provide transport service. A protection transport entity is pre-assigned as standby transport entity to provide transport service in case the working transport entity fails.

It is important to distinguish the protection topology from the topology of the network layers. Each layer has its own independent topology. See Figure 4-2.

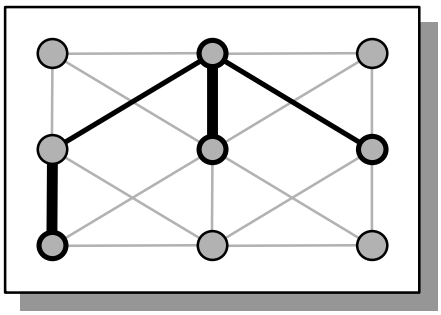


**Figure 4-2 A star topology on top of another topology**

Today only the *linear* and the *ring* topologies have proved to be of practical use. The reason for this is that protection switching is required to be a very fast mechanism so that relatively simple protection topologies are generally better suited for this purpose than complex ones. Ring protection schemes are in fact already quite complex. Note that a linear protection topology can be used on various network structures, e.g. a network structure with a ring topology.
In this report only linear protection topologies are discussed.

## 4.2 Linear Protection Switching

**Terminology**

In this report, protection switches will be called *nodes* and transport entities will be called *links*[1]. In Figure 4-3 linear protection switching is shown for the 1:N architecture (explained further on).

Two unidirectional links carrying the traffic of the same connection are called a *pair* or a bidirectional link. Two nodes in a linear protection structure together with their interconnecting working and protection links form a *protection group*. Naturally, the working and protection links should have no links or nodes in common.

[1] this is not to be confused with the link from the functional modelling method, as described in paragraph 3.2

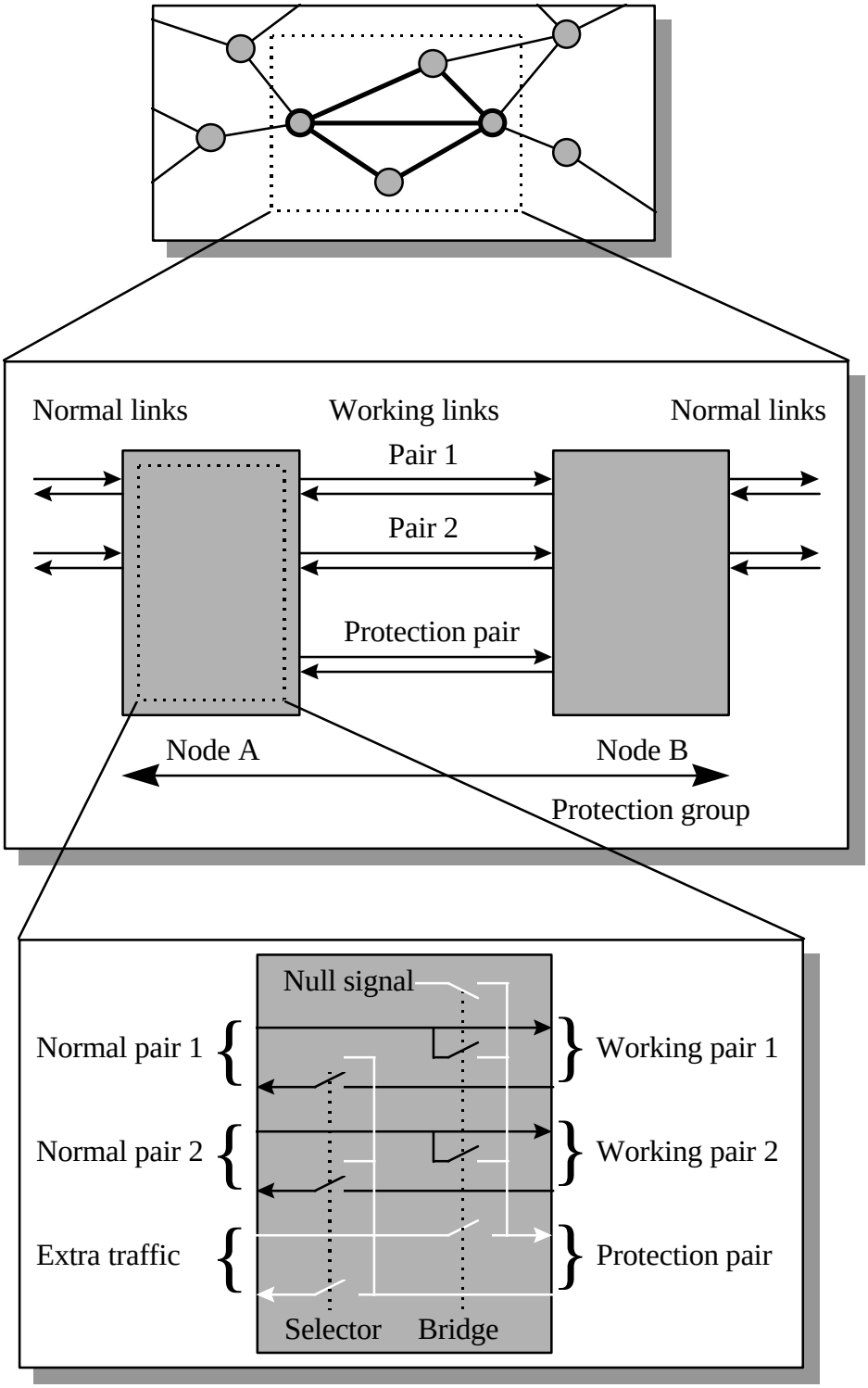



**Figure 4-3 Linear Protection Switching**

The links that carry traffic to and from the protection group, connecting the group to the rest of the network, are called *normal links*. In a non-fail situation the traffic from the normal links are transported over the corresponding working links. In case of a failure, a working link is replaced by the protection link.

This is done by *bridging* the traffic of the corresponding normal link to the protection-link at the source-side (considering the direction of the traffic) and *selecting* the traffic of the protection link and feeding it to the correct normal link at the sink-side.

Note that links are normally bi-directional if not stated otherwise. A bi-directional link can also be seen as two unidirectional links.

To make the protection switch settings more clear, the visualization as in Figure 4-4 is used.

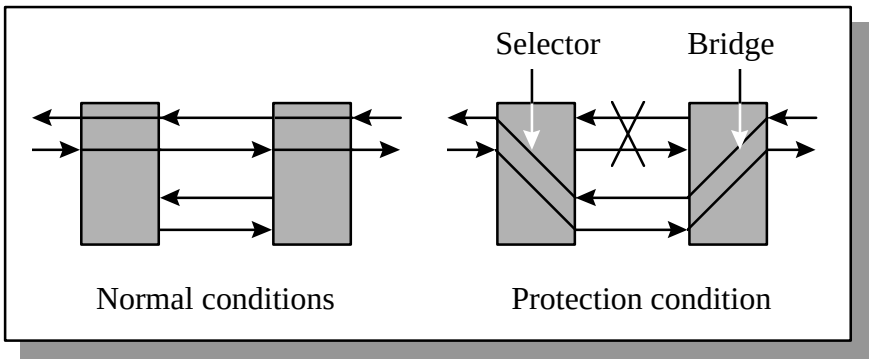



**Figure 4-4 A simpler way to visualize Linear Protection Switching**

The *bridge* switches (‘bridges’) the traffic of one of the incoming normal links over the protection link. Note that the bridged traffic is also fed to the corresponding working link, so *bridging* copies the traffic and feeds it to the protection link.
The *selector* switches (‘selects’) the traffic of the protection link to one of the normal links. If a normal is not *selected*, than it is fed with its corresponding working link.

The make specifications more readable, the links are numbered. In a protection group with N normal links and thus also N working links, the following scheme is applied:
- the protection link is numbered 0
- the normal links and their corresponding working links are numbered 1 to N

Now we can indicate which link is bridged and selected by specifying their number. A situation in which the link 1 is bridged and link 2 is selected is called a *mismatch*.
If no link is bridged, the bridge is said to be *released* and the protection link is carrying an *null signal* (an empty signal). If no link is selected than the selector is said to be *released* and all normal links are connected to their corresponding working links.

**Architecture type**

The architecture type can be either dedicated or shared.

A *dedicated* protection scheme provides a protection link that is dedicated to the protection of a single working link. This is denoted as 1+1 protection. The normal link is said to be permanently bridged to protection.

A *shared* protection scheme provides a protection link that is shared amongst more than one working links. Obviously this scheme can only protect against one failing working link. This is denoted as 1:N protection.

(A third protection scheme is M:N, i.e. there are M protection links to protect N working links. This scheme is conceptually the same as 1:N and because it is of not so much practical interest it is not discussed here)

One property of the 1:N architecture is, that it is also capable to transport ‘extra traffic’. Under non-failure conditions, the protection link is available and is used to carry extra traffic. This traffic is removed if any failure occurs. This feature can be quite profitable for the network operator if there is most of the time no failure. With respect the indicated numbering scheme, the extra-traffic link has the number N+1.

In Figure 4-5, the difference between the 1:N (N=1) architecture and the 1+1 architecture is illustrated. In the 1+1 architecture no bridge is needed (the normal link is permanently bridged) and extra traffic is not possible. This renders a much simpler structure than the 1:N architecture.

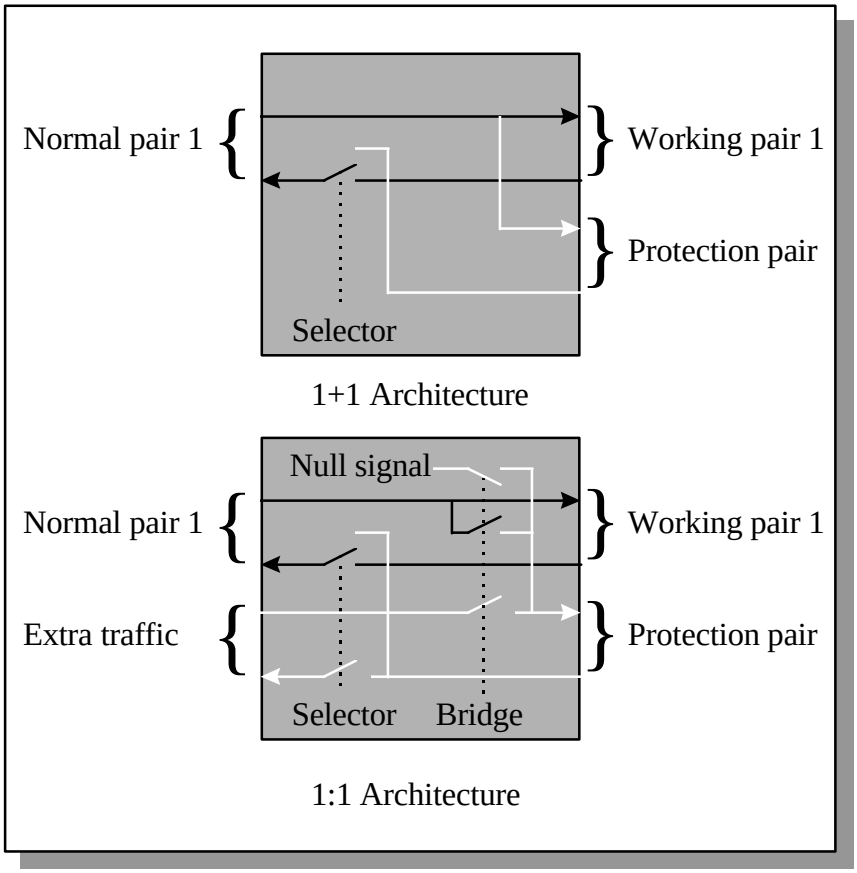



**Figure 4-5 Linear Protection Architecture Types**

Comparison:
- The 1:N scheme (N>1) is offering a lower protection level than the 1+1 scheme, because one protection link has to be shared with more working links.
- The 1+1 scheme is simpler that the 1:N scheme, because the bridge doesn't have to be controlled and consequently doesn't have to match the selector's setting at the other node.
- The 1+1 scheme has a higher cost, because every working link is protected by a protection link. The 1:N scheme is more cost-efficient, especially if extra-traffic is allowed.

A special case of the 1:N scheme is 1:1 (N=1). Although it resembles the 1+1 scheme as it also needs one protection link for a working link, it has the capability to carry extra-traffic.

An important difference between the 1+1 and 1:N schemes is that the 1:N scheme requires a communication (telemetry) channel between the two ends of the protection link to coordinate the protection switching operations and thus requires a protocol.

**Switching type**

The switching type can be unidirectional or bi-directional. *Unidirectional* switching is only concerning protection switching in the direction of the failed working link. Of course there is also protection switching in the other direction, but they are not coupled.
In *bi-directional* switching these 'unidirectional switches' are coupled. This means that if a working link fails in one direction, then both this and the opposite direction of the same working link are switched over protection.

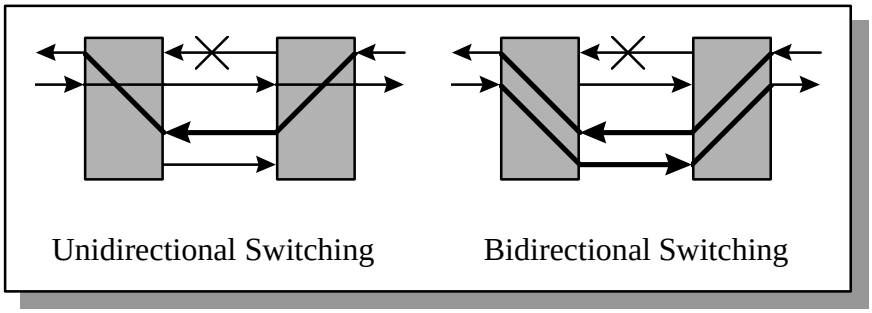



**Figure 4-6 Linear Protection Switching Type**

An important difference between the two switching types is that bi-directional switching requires a communication (telemetry) channel to coordinate the protection switching operations.

**Operation type**

The operation mode can either be revertive or non-revertive. In the revertive mode the traffic carried by the protection link as switched back to the working link once the failure is cleared. In the non-revertive mode the traffic is not switched back and remains over the same link as in the previous failure situation.

In the revertive mode a wait-to-restore (WTR) timer is introduced, to reduce unnecessary switching in situations where the condition of a link is changing relatively quick. Once the failure is (or appears to be) cleared, a timer is started. Until the timer reaches the WTR-time,

the traffic remains carried over the protection link. After this time, the traffic is switched back to the working link. In case a failure occurs within the wait-to-restore period, the WTR timer is cleared, the previous link is released and the appropriate link is switched over protection.

The non-revertive mode is only applicable in the 1+1 case. It is used to reduce switching operations. If the normal link is switched over protection, it is only switched (back) to the working link, if protection has a worse condition than the working link.

## 4.3 Linear Protection Switching in SDH

In this paragraph the details and particularities of linear protection switching in the SDH network are discussed. Some do not apply in SONET networks and many not in ATM networks

**Automatic Protection Switch channel**

As mentioned in the previous section, both the 1:N protection scheme and the bi-directional switching type require a communication (telemetry) channel to coordinate the protection switching operations. In SDH such a channel is called *APS*[1] *channel*. The coordination is handled by the *APS protocol*, which is discussed in the next paragraph.

To accommodate an APS channel on network links, some bandwidth must be reserved in those links. How this bandwidth is reserved is different for each network (layer).

Within SDH, an APS channel is defined in the multiplex section overhead (MSOH). So this applies to the multiplex section layer. The specified APS fields are coded into two bytes that are send in each MS frame (STM).

With in SDH, there is also bandwidth reserved in the path layers. In the path overhead (POH) this the K3 byte. A definition of an APS channel in this allocation is not made (yet).
Although only one byte is allocated, which cannot contain all specified APS fields together, it is still possible to define an APS channel, making use of a *multi-frame* method. The purpose of method is to group several frames in a multi-frame and then to define a ‘new’ allocation of bandwidth, making use of the allocated bandwidth per frame.
When for example the APS fields need two bytes and there is only one byte allocated in a frame, a two frame multi-frame could solve the shortage.

Although an APS channel is possible on every link of the protection group, only the APS channel definition on the protection link is used.

Within the APS channel the following fields are defined:
- Request Type (RT)
- Request Signal Number (RSN)
- Local Bridge Signal Number (LBSN)
- Architecture Indication (ARCH)

[1] Automatic Protection Switching

The Request Type field transmits protection requests. They are presented in Table 4-1. The Request Signal Number and Local Bridge Signal Number fields transmit numbers. Their meanings are discusses further on this paragraph. The Architecture Indication field transmits an indication of the architecture the sender is provisioned for.

**Table 4-1 The protection switching request types**

| **Request Type / Process State** | **Name** |
|---|---|
| LO | Lock-Out of protection |
| FSw | Forced Switch |
| SF-H | Signal Fail, high priority |
| SF-L | Signal Fail, high priority |
| SD-H | Signal Degrade, low priority |
| SD-L | Signal Degrade, low priority |
| MSw | Manual Switch |
| WTR | Wait-To-Restore |
| EXER | Exercise |
| RR | Reverse Request |
| DNR | Do Not Revert |
| NR | No Request |
| INV | Invalid request |

For SDH, several protection switching events are defined. Two kinds of these events are the changing of the link conditions and the reception of external commands.

**Link conditions**

The *link condition* is a protection switching term for the status of a link on which a protection switching action can occur. There are two kinds of *link condition*s:

- **Signal Fail (SF)** A hard failure caused by, e.g.:
  - loss of signal
  - loss of frame
  - MS AIS (Alarm Indication Signal)
- **Signal Degrade (SD)** A soft failure caused by a performance degradation detected via error detection code violation monitoring

Normally in protection switching, a link with a signal fail condition has higher priority than a link with signal degrade condition. When an a *link condition* changes, a *signal request* occurs within the protection process.

**External commands**

An *external command* can be given to a protection group by the network management via the ECC channel (described in paragraph 3.3). The commands are:

| | |
|---|---|
| **Lock-out of protection (LO)** | requests to deny all signals to the protection link. This disables the APS protocol and is needed to provision a protection group. |
| **Forced Switch #i (FSw-#i)** | requests to switch signal #i to protection. This command has higher priority than the *link condition*s. |
| **Manual Switch #i (MSw-#i)** | requests to switch signal #i to protection. This command has lower priority than the *link condition* and is removed if any signal request occurs. |
| **Exercise #i (EXER-#i)** | request for an exercise to check responses on APS bytes for normal signal #i. The switch is not actually completed. |
| **Clear** | clear any switch commands listed above. |

**Protection process state**

When the protection process has a SD, SF or an external command, which is not of higher priority than the remote request in the bidirectional mode, then the corresponding request is the state of the process. In any other case the process is in one of the following states:

| | |
|---|---|
| **No Request (NR)** | There is no local request and no remote request to react to. |
| **Reverse Request (RR)** | Only applicable in the bidirectional mode:<br>The remote request is of higher priority than the local request. |
| **Wait-To-Restore (WTR)** | Only applicable in the revertive mode:<br>The failed link has recovered, but the selection is kept for a specific period. |
| **Do-Not-Revert (DNR)** | Only applicable in the non-revertive mode:<br>The failed link has recovered, but the selection is kept until a new request presents. |

**Trail and Subnetwork Connection protection**

As discussed in paragraph 3.2, transport entities can be distinguished in trails and connections. On both types of transport entities, linear protection can be applied.

They are called:

- **Trail protection** protects one or more trails with a protection trail
- **SNC protection (SNCP)** protection one or more SNCs with a SNC

When, for example, a trail starts and ends in different operator domains (as was shown in Figure 3-7), then each operator would want to protect only that part that runs within his own domain. SNC protection makes that possible.

Because a SNC is not terminated, there is no information available to monitor the status of the SNC. However, there some monitor strategies to solve this problem:

| | |
|---|---|
| *Inherent monitoring (SNC/I)* | the status of the server link can be used to initiate protection switching. |
| *Non-intrusive monitoring (SNC/N)* | the subnetwork connection is directly monitored by use of listen-only monitoring. |
| *Intrusive monitoring* | the original trail is broken and a test trail is introduced, that extends over the SNC. All information can be monitored directly, but this is of no practical interest, because it interferes with the transport. |
| *Sublayer monitoring (SNC/S)* | A sublayer is introduced, which overwrites a portion of the original trail's capacity, such that the SNC can be monitored directly, by a trail in the sublayer. |

Another problem that results from the absence of termination of the SNC is the inability to determine where a failure has occurred within the subnetwork. It could also have happened before (upstream) the subnetwork. This problem is addressed in the following section.

**Hold-off timer**

The absence of termination in SNCP leads to some problems with the interworking of protection schemes. Some situations are shown in Figure 4-7:

| | |
|---|---|
| **Staggered protection** | If a failure occurs within a SNCP group which is a part of one of the connections of another SNCP group, the latter will also detect this failure. |
| **Partitioned protection** | If a failure occurs within a SNCP group, all the downstream SNCP groups will also report this failure. |
| **Differential delay compensation.** | If a failure occurs before (upstream) a SNCP group and one of its connections reports the failure sooner than the other, then a needless switch would be initiated. |

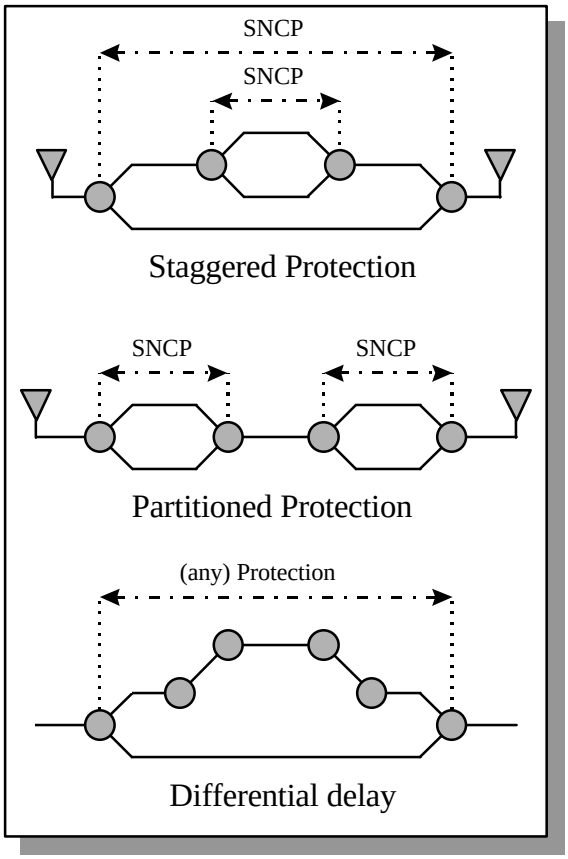



**Figure 4-7 Interworking of protection schemes**

A method to deal with these problems is the use of a hold-off timer. The hold-off time is defined as the time between declaration of signal degrade or signal fail, and the initialization of the protection switching algorithm.

In the staggered protection scheme, the surrounding SNCP should have a hold-off time greater than the time it takes for the embedded SNCP to switch.

The partitioned protection problem is presumably solved, if all SNC groups use revertive switching and if the working/protection link assignment is chosen correctly. One observation is that if a link fails in a upstream SNC group, then both links in the a downstream SNC group receive signal fail, so no switching is needed.

The differential delay problem can be solved, if the hold-off time is greater than the maximum transport delay difference between the connections.

## 4.4 Automatic Protection Switch Protocol

The APS protocol coordinates the protection switching operation. It is also referred to as the Multiplex Section Protection (MSP) protocol, because the APS channel is only defined in the MS layer.

In [Holz91] a protocol is defined as:

*'Sets of rules that govern the interaction of concurrent processes in distributed systems'*

Clearly, the protection nodes are distributed and their processes run concurrently, though synchronized by the APS messages. The counterpart of concurrent execution is sequential execution, where only one process executes at a specific time.

With the APS channel it is possible to exchange information between the protection switching nodes. This done by sending *APS messages*. APS messages are send over the APS channel. If an APS message differs from the prior message, then it initiates a request.

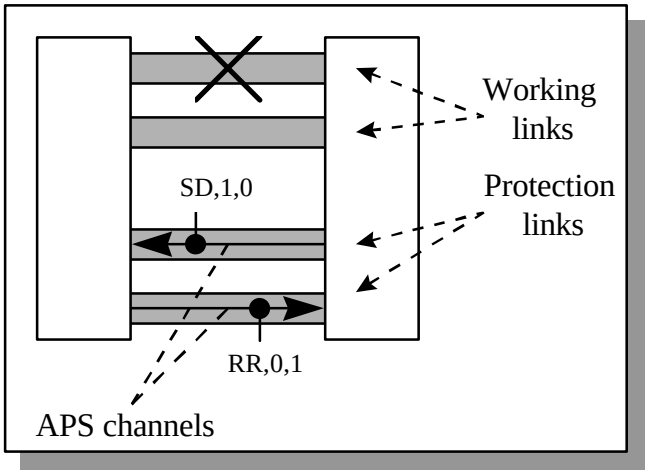



**Figure 4-8 APS messages**

As explained in paragraph 4.3, there are four fields defined in the APS channel. The architecture field is constant during the protection switching operation. The other field are used in the protocol. Together they form the APS message:

APS = RT,RSN,LBSN

For example: in a 1:N case, if the request is SF on signal 1, and the current local bridge selection is 0, then the send APS message would be: SF,1,0

The request type and the request signal number together form a *request*.

The *request type* is used in the bi-directional scheme to compare the local request against the remote request. It is received as the *remote request type*. Depending on the priorities of these two request, a node can send an *reverse request* (RR) message to indicate it accepted the request or it can send it's local request on which the remote node will reply Reverse Request.

The *request signal number* is used to indicate which signal number is to be switched over protection. It is received as the *remote request signal number*. It forms, together with the request type, a request. If the protocol is stable, then this signal is bridged by the node that received the request.

The *local bridge signal number* is used to reflect the current bridge selection of a node. It is received as the *remote bridge signal number*. It is used by the remote node to determine whether it should select a signal.

It only makes sense to select an signal if on the remote node the same signal number is bridged. If the remote bridge signal number is different from the local request signal number, the selector is temporary released. This is called temporary signal number mismatch.

Now the general operation of the APS protocol is discussed.

The purpose of the APS protocol is different for the two mentioned cases:

| | |
|---|---|
| **1:N architecture** | The signal that is to be selected at one node, must be bridged at the other node. With use of the APS protocol, the node that is to select a signal controls the remote bridge. |
| **Bi-directional switching** | The selector at one node must have the same signal as the selector at the other node. |

In the 1:N bidirectional case, both mechanisms are used.

Note that the APS channel is facilitated on the protection link. If the protection link fails, indicated by SF/0 (Signal Fail on link 0), then also the APS channel fails and the APS protocol cannot work properly. The failure can be in one direction and the other direction can be operational. This implies that protection switching is still possible in the operational direction, but it can not be coordinated. This situation is only applicable in the 1:N unidirectional case.

So, we distinct two cases in linear protection switching in SDH:

- the case that the protection is **not** failing (but it can degrade, SD/0)
- the case of a local SF/0 situation or of a remote SF/0 request as received in an APS message

Off course also in the SF/0 case, there must be exactly specified what the behaviour is. Because of the fundamental difference between the cases, it is logical that the specification of the two cases are described separately.

The APS protocol is illustrated in an example.

> The protection type in this example is 1:N bi-directional revertive, with N=2. In Figure 4-9 the APS messages are shown in a message sequence chart (MSC). The downward direction corresponds to increasing time.
> Initially there is no failing condition and both nodes are sending ‘NR,0,0’ messages continuously.
> When node B detects a failure on link 1, it sends a ‘SF,1,0’ message, indicating a request for link 1 with SF priority.
> Node A has no higher priority requests and responds with ‘RR,1,1’, indicating it is honoring the request and has set its bridge to 1.
> Node B reacts on the reception of that message by selecting link 1 and sending a ‘SD,1,1’ message, indicating it has also set its bridge to 1.
> Node A then also selects link 1 and the protection switch sequence is complete.

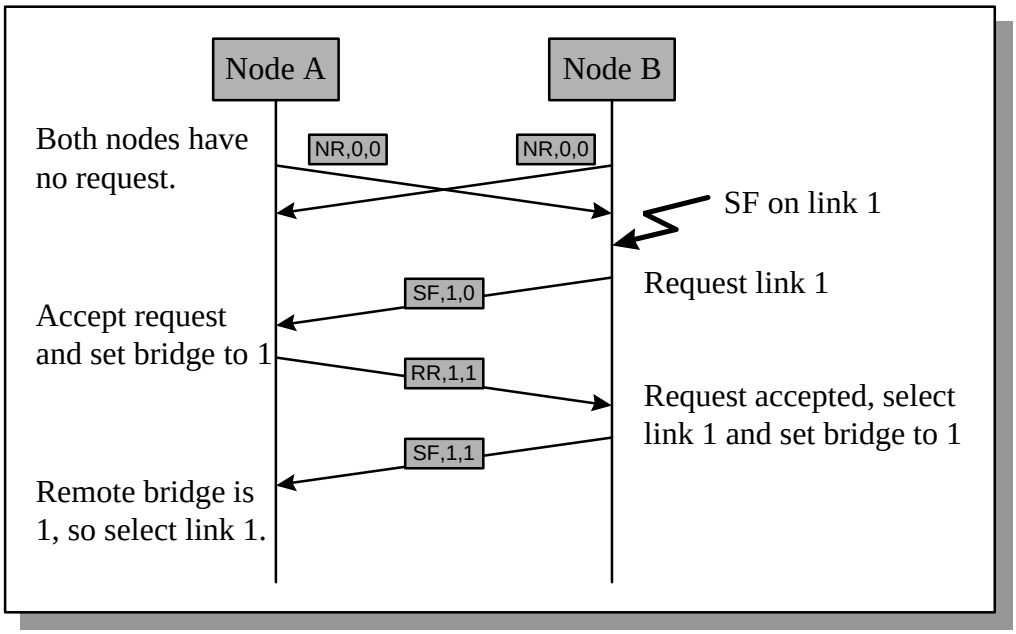



**Figure 4-9 Message flow in the APS protocol**

## 4.5 The ETSI ETS 300-417 Linear Protection Switching Specifications

The linear protection switching specification in the ETSI ETS 300-417 standard attempts to capture the behaviour of all the discussed linear protection modes. The specification resides in annex A of part 3-1 of the standard and is called ‘Generic specification of linear protection switching operation’. It is mainly based on the ITU-T Recommendation G.783 and it attempts to formalize the protection process specification to remove ambiguities present in G.783.

The specification is enclosed in annex A. An overview of the specification is presented here.

In general, the specification tries specify the behaviour of the protection switching node for each event it can have. To achieve this specification, several aspects are introduced. The definition of subprocesses, variables and states.

In both protection nodes the protection switching is accomplished by the protection process. This process is divided into subprocesses as shown in Figure 4-10:

| | |
|---|---|
| **Signal Request** | converts the *link condition*s into a (signal) request type |
| **External Request** | converts the external commands into a (external) request type |
| **Local Request Priority** | determines the highest priority local request |
| **APS Interpretation** | converts the APS message into a (remote) request type |
| **Global Request Priority** | determines the highest (global) request type comparing the local and remote requests |
| **Local Bridge Control** | determines which of the signal is bridged to protection |
| **Local Selector Control** | determines which of the signal is selected from protection |
| **APS Generation** | converts the global request and the local bridge signal number into an APS signal. |

The *local request* of a node is the highest priority request when the signal requests and external request are compared. The *global request* is the highest priority request when the local and the remote request are compared. The *remote request* is the global request of the other node, received via the APS channel.

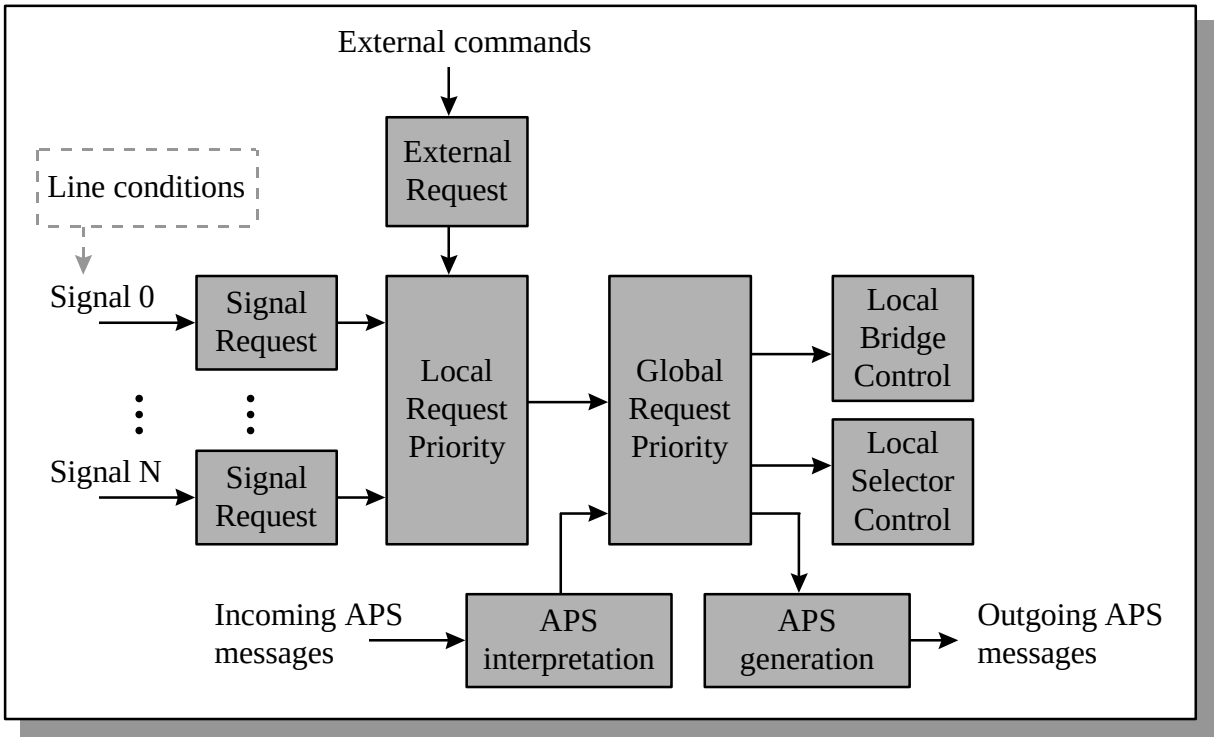



**Figure 4-10 The subdivision of the protection process**

The division in subprocesses is made on the following observations.

Firstly, the protection process reacts on events. The three kinds of events are:
- a changing link condition
- an external command
- an incoming APS message.

Secondly, the protection process examines the local situation separately from the global situation, which is the result of the comparison between the local and the remote situations.

And finally, the protection process has a state, as shown in Table 4-1 in paragraph 4.3.

The subprocesses communicate via variables. These variables are keeping data like: request type (RT) and request signal number (RSN) for each processes.

The subprocesses themselves are specified in pseudocode. A fragment of the pseudocode in the specification is shown in Figure 4-11. It is from the local request priority process, and represents a loop in which the variables LRTnew, LRSNnew and LRsource are updated if the condition of the ‘if’ statement holds.

```
1:   for i==0 to n
2:   do                                    {find highest priority local request active}
3:          if (LRTnew < SRT/i)
4:          then    LRTnew=SRT/i
5:                  LRSNnew=i
6:                  LRsource=signal
7:          fi
8:   od
```

**Figure 4-11 A fragment of the ETSI specifications pseudocode**

Pseudocode resembles the syntaxis of an imperative language like C or Pascal. It should be noted, that pseudocode is not a formal language and can contain many ambiguities, for example when ‘else’ is not specified in an ‘if-then’ statement.

In the specification every request is assigned a priority. The priority table is repeated here in Table 4-2. This table is used in the Local Request and in the Global Request Priority processes.

**Table 4-2 Request type priority**

| Priority | Request Type | Source |
|---|---|---|
| highest | LO | External command |
| \| | FSw | External command |
| \| | SF-H | Link condition |
| \| | SF-L | Link condition |
| \| | SD-H | Link condition |
| \| | SD-L | Link condition |
| \| | MSw | External command |
| \| | WTR | Protection state |
| \| | EXER | External command |
| \| | RR | Protection state |
| \| | DNR | Protection state |
| \| | NR | Protection state |
| lowest | INV | Invalid state |

‘INV’ is not an internal state to represent an invalid received request. It is not used in the specification however.

As mentioned in paragraph 4.4, the APS channel is facilitated on the protection link. This implies that a SF on protection (SF/0) is fatal for the protection protocol. In the ETSI specification, this is expressed as the assignment of a higher priority of SF/0 than FSw. Note that the external command ‘lockout of protection’ is always of highest priority, even in a failing protection situation.

# 5 System Verification

Verifying system specifications call for methods and tools. The investigation of such methods and tools is presented in this chapter. First an introduction in system verification is presented and a verification tool investigation is performed (5.1). Then the verification tool SPIN (5.2) and its modelling language PROMELA (5.3) are introduced. In 5.4 the practical and theoretical simulation and validation operation of SPIN is discussed. Finally the approach of modelling the ETSI specification in PROMELA is discussed (5.5).

## 5.1 Introduction in System Verification

Verifying system specifications means testing it to have absolute correct behaviour in all possible situation that can occur, which requires the specification of correct behaviour.

In principal systems can be verified manually, but for larger and more complex systems this is becoming increasingly difficult. This is especially true for concurrent systems, like computer and telecommunication systems and protocols. An automated verification method is a must. Correctness of protocols is discussed [Wal91] and more extensive in [Holz91].

In order to verify automatically, the use of the computer is needed and for that the system has to be described formally. The formal description of the system requires a formal language, with both a formal syntax and a formal semantics. A formal syntaxis describes the model in a precise and unambiguous way. A formal semantics assigns a precise mathematical meaning to each of the modelling primitives. See [PV97] for more information on this subject.

In addition to the formal modelling, a tool is needed that can execute this model and determine its consistency and correctness. Generally, verification tools convert the system model into an automaton or finite state machine. If the systems is bounded, the automaton is finite. Most of the systems of interest are bounded.

Besides the description of the behaviour, the verification also needs a description of the correct behaviour. These are also called correctness requirements. If all possible behaviour of the system model satisfies these requirements, the model is said to be correct and the verification is successful.

Properties of verification languages are:

- **Concurrency** Processes run unrelated to each other (this concept is always needed in telecommunication systems)
- **Synchronous / Asynchronous communication** Transmission and reception of signals between processes are at the same time or can be at different times. The latter requires the concept queue.
- **Deterministic / Non-deterministic** Identical stimuli to a system always invoke the same behaviour or may invoke different behaviour.
- **Time concept** Physical or logical time can be modelled or only the order of events can be modelled.

Furthermore, it is important that the language is understandable and intuitive and that the verification tool is easy to use.

For the verification of the ETSI protection switching specifications some languages and tools were investigated. A short summary is presented in Table 5-1.

**Table 5-1 Modelling languages and tool description**

| Language | Tool | Opinion |
|---|---|---|
| PROMELA | SPIN | The language is very understandable, it supports all mentioned properties. Time can only be modelled in a limited way.<br>The tool is easy to use and validation power is good. |
| VHDL[1] | Speedchart, FormalCheck | The language can be used for verification. It supports most of the mentioned properties.<br>Speedchart supports entering design in multiple ways (among other things, entering a FSM[2]).<br>FormalCheck can perform validations on VHDL models. |
| C++ | (home-made) | The language is broadly used and understood. It supports none of the mentioned properties, but special libraries and methods can be developed to catch up with that. |
| SDL[3] | SDT | The language is standardized by the ITU and has a broad application. It supports all mentioned features.<br>The tool SDT is capable of validating SDL systems. |

These language and tools have been investigated:

**SDL & SDT**

This tool is widely used within Lucent for system specification. In SDT, SDL diagrams can be entered, simulated and translated to C code, and also validations can be performed. But a combination of SDL semantics and the SDT implementation makes this tool useless in cases where a number of equal components are interconnected. This is e.g. the case when a protocol between a number of network elements is investigated.

**VHDL & FormalCheck**

One reason to use VHDL as modelling language could be, that the verified model can easily be used in hardware design. However, the actual benefit of this feature is still to be investigated. The modelling power of VHDL is bounded by its semantic domain. A suitable state space

[1] VHSIC Hardware Description Language, VHSIC is an acronym for Very High Speed Integrated Circuits
[2] Finite State Machine
[3] System Description Language

reduction mechanism is not provided in FormalCheck. Asynchronous communication must be modelled explicitly, as well as non-determinism.
In my opinion, VHDL is not a good description language for formal checking, because too much compromises must be made.

**C++**

A reason to use C++ as modelling language could be, that the verified model can easily be used in software design. C++ is not at all a modelling language and does not support any of the specified properties. The need to build all the required adjustments and tools to make it suitable for specification and verification, is the reason that it will not be chosen.

**SPIN**

The choice of language and tool for verifying the ETSI specification is SPIN. Its language is very clear and the validation power is very good. The only drawback is that both the input and output is only textual. SPIN and its language are discussed in the next paragraphs.

## 5.2 The Model Checker SPIN

SPIN is a software package that supports the formal verification of concurrent systems. The software was developed at Bell Labs in the formal methods and verification group.

SPIN has been used to trace logical design errors in distributed systems design, such as operating systems, data communications protocols, switching systems, concurrent algorithms, railway signaling protocols, etc. The tool checks the logical consistency of a specification. It reports on deadlocks, unspecified receptions, flags incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes.

To verify a design, a formal model is built using PROMELA[1], SPIN's input language. SPIN is an abbreviation of Simple PROMELA Interpreter. PROMELA is a non-deterministic language. It contains the primitives for specifying asynchronous (buffered) message passing via channels, with arbitrary numbers of message parameters. It also allows for the specification of synchronous message passing systems (rendezvous). Mixed systems, using both synchronous and asynchronous communications, are also supported.

The language can model dynamically expanding and shrinking systems: new processes and message channels can be created and deleted on the fly. Message channel identifiers can be passed from one process to another in messages.

Correctness properties can be specified as standard system or process invariants (using assertions), or as general linear temporal logic requirements (LTL).

[1] Process Meta Language

SPIN can be used in three basic modes:

- as a **protocol simulator**, allowing for rapid prototyping with a random, guided, or interactive simulations
- as an **exhaustive state space analyzer**, capable of rigorously proving the validity of user specified correctness requirements
- as a **bit-state space analyzer** that can validate even very large protocol systems with maximal coverage of the state space (a proof approximation technique).

The SPIN software is written in ANSI standard C, and is portable across all versions of the UNIX operating system. It can also be compiled to run on any standard PC running a Windows95 operating system.

## 5.3 The Modelling Language PROMELA

PROMELA is a validation modelling language. It provides a vehicle for making abstractions of protocols (or distributed systems in general) that suppress details that are unrelated to process interaction. PROMELA programs consist or processes, message channels and variables. Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, channels and global variables define the environment in which the processes run.

Here a few details of the PROMELA language will be shown. For a more complete description, one is referred to the annex C. The PROMELA language resembles the C language in many (notational) aspects.

**concurrency**

To describe a protocol or a distributed system, the concept of *concurrency* is needed. In PROMELA the processes are running concurrently and are communicating through channels and global variables. Unless a synchronization mechanism between processes is implemented, one cannot assume anything on the relative speeds of the processes.

One way to affect this concurrency, is to use the `atomic` keyword. If a sequence of statements is enclosed and declared atomic, then this sequence is considered one indivisible unit, non-interleaved with other processes. If a statement inside an atomic sequence is unexecutable the process will get blocked, but if it is executable again it will continue atomically.

**executability**

An important aspect of PROMELA is *executability*. In PROMELA there is no difference between conditions and statements, even isolated boolean conditions can be used as statements. The execution of every statement is conditional on its executability. Statements are either executable or blocked. The executability is the basic means of synchronization. A process can wait for an event to happen by waiting to become executable. For instance, instead of writing a busy wait loop:

```
while (a != b)
     skip
```

one can achieve the same effect in PROMELA with the statement

```
(a==b)
```

A condition can only be executed (passed) when it holds. If the condition does not hold, execution blocks until it does.

If a process is blocked, it can only execute again if its blocking cause is removed by another process. When every process is blocked, that is no statement in the entire model is executable, then a *timeout* occurs. This a special feature of PROMELA, making it possible to model an escape from the normal program flow.
A special statement, called `timeout` is not executable while other statements are executable. When a timeout occurs, the timeout statements becomes executable. The timeout statement is typically used as a guard in a selection or repetition structure, to model a protocol timeout.

The following code sends a request to inform another process that is ready to receive packets, it acknowledges every received packet and sends more requests when it takes to long:

```
Send_Request ;
do
:: Receive_packet -> Send_Ack
:: timeout -> Send_Request
od ;
```

The `do::od` repetition statement is discussed further on.

**processes**

The behavior of a process is defined in a ‘proctype’ declaration. It must be instantiated through the `run` operator. Initially, one special process will be executed: the ‘init’ process, that must be declared in every PROMELA specification.

**channels**

Message channels are used to model the transfer of data from one process to another. A message declaration defines a name, a channel length and the message type. For instance:

```
chan qname = [16] of { short }
```

The channel qname can store up to 16 messages of type short. The statement

```
qname!expr
```

sends the value of expression expr to the channel, that is: it appends the value to the tail of the channel (it can be seen as a queue). The statement

```
qname?msg
```

receives the message, it retrieves it from the head of the channel.

The send operation is executable only when the channel is not full. The receive operation, similarly, is only executable when the channel is not empty.

In the above example, a channel was declared with a length greater than zero. This makes the communication between the processes *asynchronous*. Which means that the sending of a message from one process can be at another time than the receiving of the message by another process.

If the length of the channel is chosen zero, then the communication is *synchronous*. In this case the channel cannot store messages and the send and receive operation must occur at the same time. This is also called *rendezvous* communication. With this construction a semaphore can be build, for example blocking process A, until process B has reached a point.

**non-determinism**

Another special aspect of the PROMELA language is *non-determinism*. This is reflected in the selection statement.
An example of the use of the selection statement is explained here:

```
if
:: (a != b) -> option1
:: (a == b) -> option2
fi
```

The selection contains one or more execution sequences, each preceded by a double colon. Only one sequence from the list will be executed. A sequence can be selected only if its first statement is executable. The first statement is therefore called a *guard*.

In the above example the guards are mutually exclusive, leading to a deterministic choice. If more than one guard is executable, one of the corresponding sequences is selected *non-deterministically*. If all guards are unexecutable, then the selection statement itself is unexecutable.

The repetition statement is functionally the same, expect that it does not execute one sequence, but it keeps executing sequences. It is possible to escape from this repetition by use of the `break` or the `goto` statement:

```
do
:: (a<10)    -> a++
:: else      -> break
od ;
```

**simulation output**

It is possible (during a simulation) to write lines of text on the console in which SPIN is running, with to `printf` statement. This provides a means to write specific information about a certain state.

## 5.4 Simulating and Validating with SPIN

**SPIN program control**

SPIN is a commandline tool running in a UNIX shell or at a DOS commandshell. In the UNIX environment the program is called ‘spin’, in the DOS environment ‘SPIN.EXE’. SPIN also needs a standard C preprocessor available (CPP). SPIN uses CPP to preprocess the PROMELA model. This enables the designer to use any C preprocessor command.

The typical program options are discussed. To simulate a PROMELA model in a file called ‘protocol.pro’ the following command is issued, where ‘>’ is the command prompt:

```
>spin protocol.pro
```

Several flags can be applied to change the SPIN program behaviour.

To validate a model, the following commands must be issued:

```
>spin -a protocol.pro
```

to create a set of files, which is the C source of a validator, tailored and optimized for the specified model. One of these files is ‘pan.c’. This file must be compiled to create an executable with:

```
>gcc -o pan pan.c
```

This generates the executable ‘pan’ (Protocol Analyzer). This program carries out the validation and report the results. To the compilation and the execution of the protocol analyzer also several options are available. One of the options controls the search strategy discussed in this paragraph. The options are not discussed here, the reader is referred to annex C for the details.

**Execution sequences**

The system behavior of a validation model is defined by all possible execution sequences it can perform. An *execution sequence* is finite, ordered set of states. A state is completely defined by the specification of all values for local and global variables, all control flow points of running processes and the contents of all message channels.

In the begin state all variables are initialized to zero, all message channels empty, only the ‘init’ process is active and the control flow point is set the first statement in the ‘init’ process. All statements that are executable in a given state, result in new states.

So the set of reachable states of a model is a *finite state machine* (FSM). A FSM is described with states (nodes) and transitions (arrows) between the states. This could be visualized as in Figure 5-1: a state tree, with extra properties: a tree in which loops and cross-connects, and therefore multiple predecessors to a node, are allowed. With the root being the begin state and

the leaves the end states. Each path from the root to the leaves or cycling in a loop is an execution sequence.

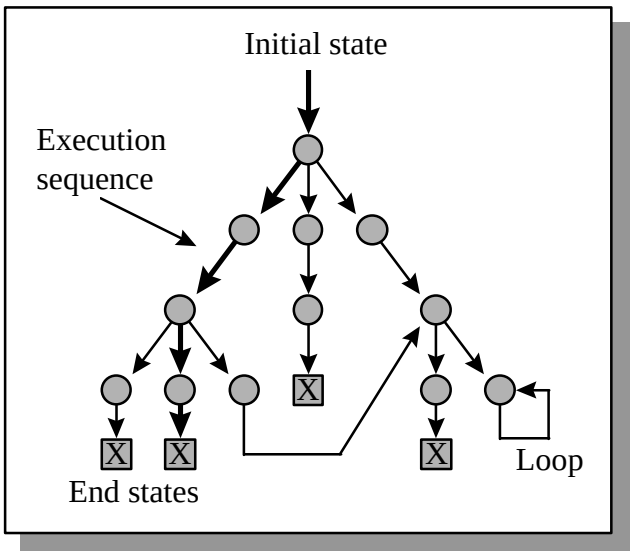



**Figure 5-1 Execution sequences**

From a PROMELA viewpoint, there are two mechanisms that result in different execution sequences:

- concurrency: the interleaving of executable statements from the running processes.
- non-determinism: the different choices that could be made at executable selection- or repetition statements.

From the SPIN viewpoint (how SPIN processes the model), these are actually the same: at a specific state, one of all executable statement is chosen.

Now the difference between simulation and validation can be made clear:

- In a *simulation*, one execution path is chosen.
- In a *validation*, each possible execution path is followed once.

With the SPIN tool, simulations are typically used to observe whether the model implements the intended behavior, before a validation is started. As mentioned before, `printf` statements can be used to write text to the console.

Validations are used to trace errors in the model. Therefore all states must be explored to check for errors. This is also called *exhaustive reachability analysis* or *state space search.*

**Correctness criteria**

To verify a model, one must not only specify the behavior, one must also specify what the correct result is. Therefore *correctness criteria* must be specified in the PROMELA model. Correctness criteria are formalized as claims about the behavior of the model. In PROMELA, claims are always formulated as behavior that is impossible, as opposed to formulation of behavior that is inevitable.

These are ways to specify correctness criteria in PROMELA:
- the **assertion**, formulated with the `assert` statement, which is a claim that a boolean condition must be satisfied at that point.
- the **end-state** label, to define proper end-states in non-terminating cycles: a intended non-terminating cycle, is not a dead-lock.
- the **progress-state** label, to claim that the labeled state must always be passed in a execution cycle. Execution sequences that violate this claim are called non-progress cycles.
- the **acceptance-state** label, to claim that the labeled state cannot be passed infinitely often in a execution cycle. Execution sequences that violate this claim are called livelocks.
- formalization of general **temporal claims**. A temporal claim defines temporal orderings of properties of states, such as ‘every state in which property P is true, is followed by a state in which property Q is true’. Such a claim is implemented in PROMELA as a ‘never’ claim.

In simulations, only the assert statements are checked. The other claims can only be checked in a validation.

One notorious error in protocols is the *dead-lock*. This is a situation, that two entities are waiting for a service from each other, making it impossible to continue execution. This situation could easily be detected with use of one or more claims. For example, by indicating that the suspect state is not a valid end-state. More advanced claims could be formulated by never-claims.

If an error is detected in a validation, a *trace* file is written, indicating the execution sequence that lead to the error. Running the simulator in the *guided* mode, the simulation is guided according to the trace (instead of making *random* choices) and the erroneous behavior can be observed.

**Validations in practice**

An important issue in validation is the enormous number of states a model on a distributed system can have. This is sometimes referred to as the *state explosion problem*. Even a modest protocol can have millions of states. This makes an complete validation in most cases impossible, but there are algorithms that can cope with this problem.

First two measures for expressing the capabilities of a reachability analysis are presented:
- *Coverage*, the search’s ability to reach states: the number of states tested divided by the number of states in the full state space.
- *Quality*, the search’s ability to find errors: the number of distinct errors found divided by the total number of errors present.

To rigorously prove the correctness of a model using an automated validation system, such as SPIN, a full state space search is needed. The coverage and the quality are both 100%. But it may not be possible. In general, when performing a task on a computer, there are the following aspects:
- The speed of the calculations, i.e. the speed of the computer
- The size of the memory in which the calculations are performed
- The time that is needed to complete all calculations

An exhaustive analysis may be undesirable, because it could take to long or could detoriate into a low-quality partial search if the memory is too small. The latter is explained: If the size of the state space is R and the maximum available memory is A, both the coverage and the search quality can only reach 100% when R<A. When R>A the coverage reduces to A/R, but the search quality is likely to be worse. This is because only an 'solid' cut of the statespace is reached and errors are likely to be distributed. See Figure 5-2.

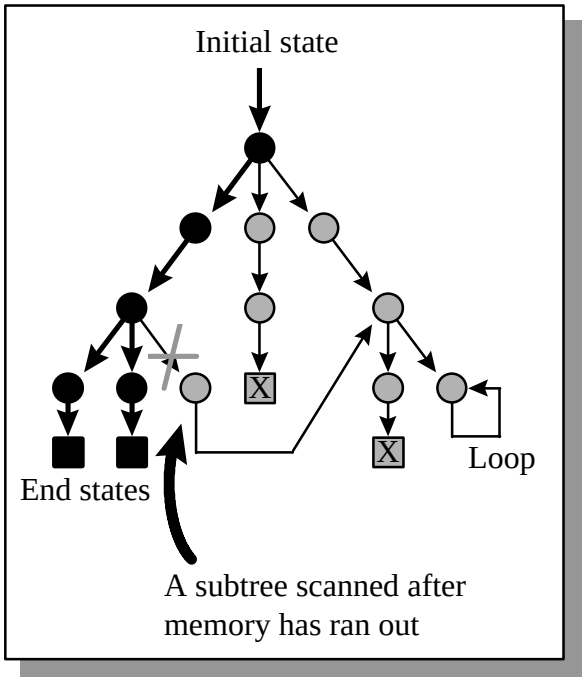



**Figure 5-2 Low search quality**

An example will illustrate the problem:
Consider a protocol with a total number of states in the order of $10^{12}$, which is a modest size protocol. If we assume, quite unrealistically, that each state can be encoded in 1 byte of memory and can be analyzed in $10^{-6}$ sec of CPU time, we would still need a machine with at least 1000 Gbytes of memory, and would need roughly 12 days to perform an exhaustive analysis.

There are two algorithms to improve the search quality in a limited memory area:
- the random simulation
- the controlled partial search

These are also called *proof approximation techniques*.

The *random simulation* technique is largely independent of the size and complexity of the system; even infinite size systems can be explored. The search is completely random and there is no record whether an execution sequence is explored before. This technique gives no measure of the coverage nor the search quality, and is therefore only used as a last resort.

A *controlled partial search* aims to achieve the best result within a limited memory area. A controlled partial search has the following objectives:

- To analyze **precisely** A states, with A=M/S. M is the amount of memory that is available for the search. S is the memory needed to store one state.
- To select these A states from the complete set of reachable states R in such a way that all major protocol functions are tested.
- To select the A states in such way that the search quality is better than the coverage A/R

**Supertrace algorithm**

The technique that SPIN uses to implement a controlled partial search is called the *supertrace algorithm* or the *bit state space analysis.* A short overview of this technique will be presented:

The supertrace technique is based on a storage technique called *hashing*. A hash function projects a value onto an arbitrary hash-value in a range 0..H-1, in such way that the same value is always projected onto the same hash-value. This is shown in Figure 5-3.
There is also a possibility that two different values project onto the same hash-value. This is called a *hash-conflict*.

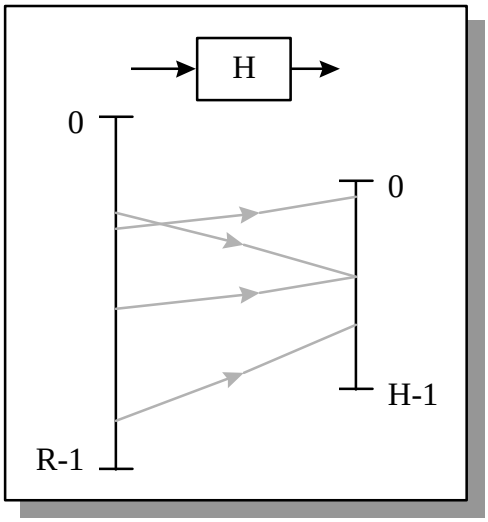



**Figure 5-3 The hash function projection**

This characteristic is used to implement the partial search algorithm. Each state is projected onto one bit of memory. The value of this bit determines whether the state should be analyzed or not. If more than one state project onto a bit, then the one analyzed first sets the bit and all the other states that project onto this bit, are not analyzed; the search subtree is pruned. This causes the search to be partial controlled. See Figure 5-4.

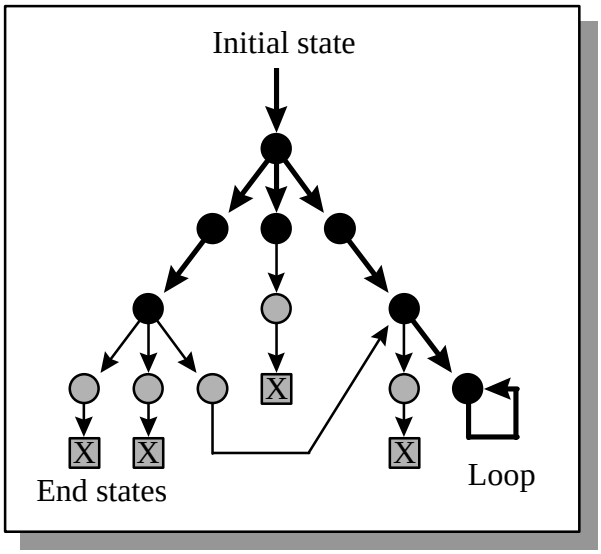



**Figure 5-4 Partial Search**

Note that the term random is somewhat ambiguous, when talking about random selections. Although the selection is arbitrary, it is also reproducible, because every simulation or validation the hash function is the same.

A indication of the coverage of a bit state space analysis is the *hash-factor*. A hash-factor near 1 implies a poor coverage, a high number (more than 100) a good coverage. A hash factor (much) greater than 100 corresponds with a coverage over 99.9%

A very convenient side-effect of the hashing technique, is that the number of hash-conflict is controllable. This can be done by specifying the size of the hash table. This corresponds to specifying the memory area available to the validation.

In Table 5-2, an example with a model with 334.151 reachable states demonstrates the correlation between the hash factor and the coverage. Note the last line: the exhaustive analysis degrades to an uncontrolled partial search with poor coverage if the memory area is to small. The coverage of the supertrace search in the same area is much better.

**Table 5-2 Correlation between Hash Factor and Coverage**

| *Search* | *Hash-factor* | *States stored* | *Hash collisions* | *Memory used* | *Coverage* |
|---|---|---|---|---|---|
| Exhaustive | - | 334.151 | 66.455 | 45.6 MB | 100.00 % |
| Supertrace | 100.9 | 332.316 | 1.835 | 9.9 MB | 99.45 % |
| Supertrace | 50.9 | 329.570 | 4.581 | 7.9 MB | 98.62 % |
| Supertrace | 25.7 | 326.310 | 7.841 | 6.9 MB | 97.65 % |
| Supertrace | 13.0 | 322.491 | 11.660 | 6.3 MB | 96.51 % |
| Exhaustive | - | 83.961 | 389.671 | 6.3 MB | 25.12 % |

## 5.5 The ETSI Specifications modelled in PROMELA

**Investigation**

Before modelling the ETSI specifications in PROMELA, an investigation is done how the process specification map into the PROMELA features. Here is a summary:

∗ The PROMELA processes run concurrently. The ETSI protection processes also run concurrently, but the subprocesses do not. The execution order between the subprocesses is always the same, modelling the subprocesses in PROMELA processes and linking them with channels would be a waste of time (both in modelling time as in simulation and validation time). Instead of this, the subprocesses are modelled as blocks of code, all belonging to one process, jumped to and from with `goto` statements.

∗ The APS channel can be modelled perfectly with PROMELA channels. But how can the changing of the link condition and the external commands be modelled? For this purpose channels are used. Each protection node monitors the condition of the incoming links. For each incoming line a channel is declared which transport NR, SD, or SF messages to model the notification of a new link condition. Now, all event are modelled as incoming messages on PROMELA channels.

∗ The protection process is idle if the APS channel is idle, that is if the same APS message is sent repeatedly. While the APS protocol is executing, messages flow over the APS channel. If the protocol has finished, the APS channel is idle again. This is modelled as follows: At the beginning of the PROMELA protection process, execution is halted until a message is received on a channel. If a message is received and if it is different from the previous message, execution is jumped to the corresponding subprocess block. After execution of all relevant code, an APS message is send and the process is blocked again, waiting for the next event.

∗ All the Pseudocode statements and data types map (sufficiently) good onto PROMELA statements and data types. In Figure 5-5 some examples of this mapping is shown.

| Pseudocode | PROMELA code equivalent |
|---|---|
| ‘if statement’ | |
| 1: if (count==last)<br>2: then count = 0<br>3: else count = count + 1<br>4: fi | 1: if<br>2: :: (count==last) -><br>3: count = 0<br>4: :: else -><br>5: count = count + 1<br>6: fi ; |
| ‘for statement’ | |
| 1: for i=0 to n<br>2: do sum = sum + number[I]<br>3: od | 1: i := 0<br>2: do<br>3: :: (i>n) -><br>4: break<br>5: :: else -><br>6: sum = sum + number[i] ;<br>7: i++<br>8: od ; |

**Figure 5-5 The mapping of the pseudocode onto PROMELA**

### Implementation

The PROMELA model of the protection switching specifications is included in annex B. Here an overview is given. In Figure 5-6 a fragment of the PROMELA code is shown that corresponds to the fragment of pseudocode in Figure 4-11.

```
1:  i = 0 ;
2:  do
3:  :: i>Nmax -> break                  /* All signals processed */
4:  :: else ->
5:         if
6:         :: (srt[i] > lrtnew) ->
7:                lrtnew = srt[i] ;
8:                lrsnnew = i ;
9:                lrsource = SOURCE_SIGNAL
10:        :: else -> skip
11:        fi ;
12:        i++ ;
13: od ;
```

**Figure 5-6 A fragment of the PROMELA model of the ETSI specifications**

The blocks that divide the process into the subprocesses map exactly onto the subprocesses described in paragraph 4.5. They are shown in Figure 5-7. The blocks are summarized in Table 5-3.

**Table 5-3 The labels of the code of the subprocesses**

| Block | Next block jumped to |
|---|---|
| waitmsg | depending on the msg to:<br>apsin,<br>extern or signalrequest |
| apsin | global |
| extern | local |
| signalrequest | local |
| local | (next) |
| global | (next) |
| bridge | (next) |
| selector | (next) |
| transmit | waitmsg |

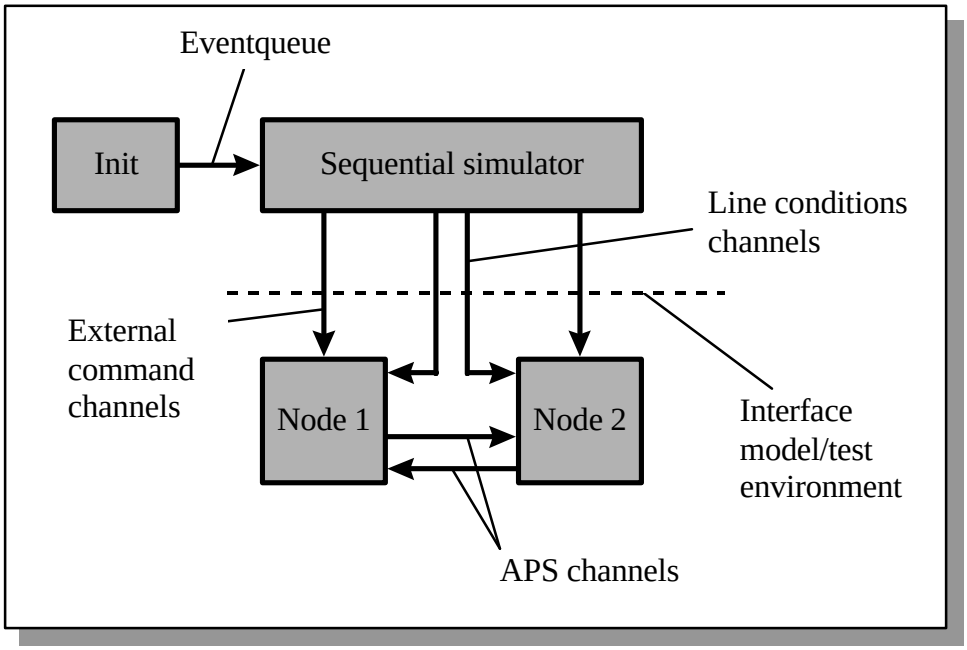



**Figure 5-7 The PROMELA model of the ETSI specifications**

Just before the transmission of the APS message in the TRANSMIT block the status is printed to console using the 'printf' command. The APS message that is send is printed, together with the current bridge and selector setting. This is enough information to make a decision whether the behaviour of the protection processes is correct.

**Eventqueue**

As mentioned in paragraph 5.3, a timeout occurs if no statement in the PROMELA model is executable. This PROMELA feature is gratefully used in this model. If the protocol has finished executing, observed by the fact that the APS channel is idle, then a timeout occurs. This timeout is used by another process in the model, called *sequential_simulator*.

This process is defined for programming convenience, it could also have been coded inside the init process. It reads events from a channel called *event_queue*, sends the event to the appropriate process and wait for a timeout to occur. When the timeout occurs, the next event is read. This is repeated until the event queue is empty, then the execution of the complete model stops.

The init process fills the event_queue with events, executes the processes and stops. The events that are inserted in the event queue are link condition and external command events plus a marking for which protection node it is meant.

When the protocol has finished executing and before the next event is read, the correctness rules are applied. Assert statements are enough to describe the correctness of the protocol. If an assertion is false, then in both the simulation and validation case the executions is stopped and the false assertion is notified.

In the simulation mode, these events are explicitly programmed in the init process. In the validation mode another process is used. A process called random fills the event queue with three events chosen from all the events that are defined.

**Modelling time in PROMELA**

The concept time is not present in PROMELA, other than the timeout condition, discussed in paragraph 5.3.

Time can be modelled in several forms, for example:
- Time is expressed in the physical time unit, i.e. seconds (milliseconds, nanoseconds, etc.),
- Time is expressed in logical units, e.g. clock ticks,
- Time is expressed in a condition, which indicates relative order to another time instant. (e.g. time A is *after* time B).

Without any time concept it is impossible to verify hold-off timing problems as discussed in paragraph 4.3. Inspired by this problem a method is designed to model time in PROMELA. The PROMELA code is supplied in annex D.

The methods uses logical units to express time. It consists of one process, a global variable ‘time’ and a set of preprocessor functions.

The process increments the global variable ‘time’ with one when a timeout occurs. Processes can check the current time and wait for a specific time by setting timers.

The only implication of the method used here, is that it only works if no other process uses the timeout condition. This is no problem if such a process can also use the method supplied time functions.

Although, the hold-off problems are not actually verified in this research, the method has been used to model hold-off timing and appeared to be sufficient.

# 6 ETSI Specification Verification

In this chapter the findings of the verification of the ETSI specifications are discussed. First an overview is given what simulation and validation results with SPIN look like (6.1). In 6.2 all findings are discussed that are clearly traceable in the specification and possible corrections are proposed. Finally in 6.3 some open issues are discussed, that need future attention.

## 6.1 Simulation and Validation Output

**Simulation output**

When simulating with SPIN, the only output is lines of text. As mentioned in paragraph 5.5, the PROMELA model of the ETSI specifications has output that contains the following information:

- the APS message that is send,
- an indication which nodes send it,
- the current bridge setting of that node,
- the current selector setting of that node.

The simulation output is shown in an example in Figure 6-1.

```
1:   # Remark:
2:   # Parameters ETSInode(1): 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
3:   # Parameters ETSInode(2): 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
4:   # |----|----|----------|----------|----|----|
5:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
6:   # |----|----|----------|----------|----|----|
7:   # |  0 |  0 | NR  ,0,0 >          |  0 |  0 |
8:   # |  0 |  0 |          < NR  ,0,0 |  0 |  0 |
9:   # |----|----|----[initialized]----|----|----|
10:  ### Node 1, command SD for signal 1
11:  # |  0 |  0 | SDL ,1,0 >          |  0 |  0 |
12:  # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
13:  # |  1 |  1 | SDL ,1,1 >          |  0 |  1 |
14:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
15:  ### Node 1, command NR for signal 1
16:  # |  1 |  1 | WTR ,1,1 >          |  1 |  1 |
17:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
18:  ### Node 1, command TO
19:  # |  0 |  1 | NR  ,0,1 >          |  1 |  1 |
20:  # |  0 |  1 |          < NR  ,0,0 |  0 |  0 |
21:  # |  0 |  0 | NR  ,0,0 >          |  0 |  0 |
22:  # |  0 |  0 |          < NR  ,0,0 |  0 |  0 |
23:  ### Node 2, command SD for signal 2
24:  # |  0 |  0 |          < SDL ,2,0 |  0 |  0 |
25:  # |  0 |  2 | RR  ,2,2 >          |  0 |  0 |
26:  # |  0 |  2 |          < SDL ,2,2 |  2 |  2 |
27:  # |  2 |  2 | RR  ,2,2 >          |  2 |  2 |
28:  ### Node 1, command SF for signal 1
29:  # |  0 |  2 | SFL ,1,2 >          |  2 |  2 |
30:  # |  0 |  2 |          < RR  ,1,1 |  0 |  1 |
31:  # |  1 |  1 | SFL ,1,1 >          |  0 |  1 |
32:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
33:  ### Node 1, command NR for signal 1
34:  # |  1 |  1 | WTR ,1,1 >          |  1 |  1 |
35:  # |  1 |  1 |          < SDL ,2,1 |  0 |  1 |
36:  # |  0 |  2 | RR  ,2,2 >          |  0 |  1 |
37:  # |  0 |  2 |          < SDL ,2,2 |  2 |  2 |
38:  # |  2 |  2 | RR  ,2,2 >          |  2 |  2 |
39:  ### Node 2, command NR for signal 2
40:  # |  2 |  2 |          < WTR ,2,2 |  2 |  2 |
41:  # |  2 |  2 | RR  ,2,2 >          |  2 |  2 |
42:  ### Node 2, command TO
43:  # |  2 |  2 |          < NR  ,0,2 |  0 |  2 |
44:  # |  0 |  0 | NR  ,0,0 >          |  0 |  2 |
45:  # |  0 |  0 |          < NR  ,0,0 |  0 |  0 |
46:  # |  0 |  0 | NR  ,0,0 >          |  0 |  0 |
47:  #processes: 4
```

**Figure 6-1 The PROMELA model simulation output**

The formatting is designed to make the output maximally readable. The creation of an event is shown on lines starting with '###'. The consecutive behaviour can be read from the lines starting with '#'. The column '`APS 1->2`' the APS messages from node 1 to node 2 are displayed and the other way around in column '`APS 2->1`'. The columns '`LSSN`' and '`LBSN`' reflect the settings of the selector and the bridge respectively for node 1 (left) and node 2 (right).
Before any event is created, both nodes are initialized (lines 7-9). When the protocol is stable, the settings can be compared to the desired behaviour, e.g. line 14.

The printed information is enough to determine errors, because there is no more than the setting of the bridge and selector and the initiating event. Information about the APS protocol behaviour is (very) convenient, but not necessary. Once an error is observed, it is often not difficult to locate the source in the model.

### Validation output

Also in the validation mode SPIN produces text as output. If a validation is run and no error is found the following output can be observed. In Figure 6-2 the result of an exhaustive search is presented.

```
1:   (SPIN Version 2.9.6 -- 20 March 1997)
2:          + Partial Order Reduction
3:
4:   Full statespace search for:
5:          never-claim           - (none specified)
6:          assertion violations  +
7:          cycle checks          - (disabled by -DSAFETY)
8:          invalid endstates     +
9:
10:  State-vector 20 byte, depth reached 13, errors: 0
11:        21 states, stored
12:         1 states, matched
13:        22 transitions (= stored+matched)
14:         1 atomic steps
15:  hash conflicts: 0 (resolved)
16:  (max size 2^18 states)
17:
18:  1.493  memory usage (Mbyte)
19:
20:  unreached in proctype proc
21:         (0 of 10 states)
22:  unreached in proctype :init:
23:         (0 of 4 states)
```

**Figure 6-2 SPIN validation output in Full statespace search**

The model is extremely small (line 11: 21 states) and only inserted here to show the typical output of a validation in full statespace search mode. As line 5 to 8 show, there is only searched for assertion violations and invalid endstates. That is because only assertion are used for the correctness rules. The latter is turned on by default.

In Figure 6-3 the result of a supertrace search is shown.

```
1:    (SPIN Version 2.9.6 -- 20 March 1997)
2:          + Partial Order Reduction
3:
4:   Bit statespace search for:
5:          never-claim           - (none specified)
6:          assertion violations  +
7:          cycle checks          - (disabled by -DSAFETY)
8:          invalid endstates     +
9:
10:  State-vector 308 byte, depth reached 1492, errors: 0
11:  2.48408e+06 states, stored
12:    486535 states, matched
13:  2.97061e+06 transitions (= stored+matched)
14:  7.35219e+06 atomic steps
15:  hash factor: 13.5078 (expected coverage: >= 98% on avg.)
16:  (max size 2^25 states)
17:
18:  Stats on memory usage (in Megabytes):
19:  775.032 equivalent memory usage for states (stored*(State-vector + overhead))
20:  4.194   memory used for hash-array (-w25)
21:  0.280   +memory used for DFS stack (-m10000)
22:  0.163   +memory used for other data structures
23:  8.873   =total actual memory usage
```

**Figure 6-3 SPIN validation output in Bit statespace search**

This model is somewhat larger (line 12: two and a half million states). For this validation a partial search is used. As shown in line 10, the state-vector size is 308 byte. If this value is multiplied with the number of states, then the equivalent memory usage is obtained as in line 19.

What is more important, is the value of the hash factor (line 15). The value of 13.5 corresponds with an estimated coverage of 98%. Although this may seem a good value, this is not satisfactory for a verification, because 2% unsearched states is large enough to worry about unseen errors. As mentioned in paragraph 5.4, a hash factor (much) greater than 100 (a estimated coverage greater than 99,9%) would be a better indication that the model is nearly fully verified. For that, another validation with other parameters is needed.

## 6.2 Verification results

When simulating and validating the ETSI specifications, several errors and findings were observed. They are discussed in this paragraph. Every subject is started with an introduction. Then the location in the ETSI pseudocode is pointed out and when possible a suggestion is done that corrects the problem.

### 6.2.1 Reverse Request is replied with Reverse Request

This error only applies to the bidirectional case.

It occurred after all failures have cleared and the last requesting node sends NR (in the revertive case after a WTR timeout, in the non-revertive case only if the last switch concerned protection): *The remote node should reply NR, but it replies RR.*

This was originally specified in the textual part of the G.783 standard, but it was not specified in the item list of this document. Through this mistake it was not specified in the ETSI standard.

The incorrect behaviour is shown in Figure 6-4. In Figure 6-5 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SD for signal 1
6:   # |  0 |  0 | SDL ,1,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
8:   # |  1 |  1 | SDL ,1,1 >          |  0 |  1 |
9:   # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
10:  ### Node 1, command NR for signal 1
11:  # |  1 |  1 | WTR ,1,1 >          |  1 |  1 |
12:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
13:  ### Node 1, command TO
14:  # |  0 |  1 | NR  ,0,1 >          |  1 |  1 |
15:  # |  0 |  1 |          < RR  ,0,0 |  0 |  0 |
16:  # |  0 |  0 | NR  ,0,0 >          |  0 |  0 |
17:  # |  0 |  0 |          < RR  ,0,0 |  0 |  0 |
```

**Figure 6-4 Reverse Request is replied with Reverse Request: Incorrect behaviour**

```
Original code (in global request priority process)

1:   Global request priority process
2:   ------------------------------
3:   if (SWtype==bi-directional) and (SRT/0!=SF) and (RRT!=RR) and
4:      [   (RRT>LRT) or
5:          ((RRT==LRT) and (GRT==RR)) or
6:          ((RRT==LRT) and (GRT!=RR) and (RRSN<LRSN))
7:      ] )
8:   then   GRT=RR
9:          GRSN=RRSN
10:  else   GRT=LRT
11:         GRSN=LRSN
12:  fi

Corrected code

1:   Global request priority process
2:   ------------------------------
3:   if (SWtype==bi-directional) and (SRT/0!=SF) and (RRT!=RR) and
4:      [   (RRT>LRT) or
5:          ((RRT==LRT) and (RRT!=NR) and (GRT==RR)) or
6:          ((RRT==LRT) and (RRT!=NR) and (GRT!=RR) and (RRSN<LRSN))
7:      ] )
8:   then   GRT=RR
9:          GRSN=RRSN
10:  else   GRT=LRT
11:         GRSN=LRSN
12:  fi
```

**Figure 6-5 Reverse Request is replied with Reverse Request: Correction in code**

### 6.2.2 Forced switch has higher priority than SF on protection

This problem only applies to the case an APS channel is used.

A signal fail request for protection should have a higher priority than a forced switch request when the APS channel is used, *but this is not specified in the code.*

The incorrect behaviour is shown in Figure 6-6. Note that an APS message can not be received when the protection link is failing. This can be observed in the figure by the fact that there is no reply from node 1 after lines 3 and 5.
In Figure 6-7 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SF for signal 0
6:   # |  0 |  0 | SFL ,0,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < RR  ,0,0 |  0 |  0 |
8:   ### Node 1, command FSw for signal 1
9:   # |  0 |  0 | FSw ,1,0 >          |  0 |  0 |
10:  # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
```

**Figure 6-6 Forced switch has higher priority than SF on protection: Incorrect behaviour**

```
Original code (in local request priority process)

1:   for i=0 to n
2:   do     if (LRTnew < SRT/i)
3:          then   LRTnew=SRT/i
4:                 LRSNnew=i
5:                 LRsource=signal
6:          fi
7:   od

Corrected code

1:   for i=0 to n
2:   do     if (LRTnew < SRT/i)
3:          then   LRTnew=SRT/i
4:                 LRSNnew=i
5:                 LRsource=signal
6:          fi
7:   od
8:   if (LRTnew==FSw) and (SRT/0==SF) and (APSmode=true)
9:   then   LRTnew=SF
10:         LRSNnew=0
11:         LRsource=signal
12:  fi
```

**Figure 6-7 Forced switch has higher priority than SF on protection: Correction in code**

### 6.2.3 Forced switch is not removed at SF on protection

This problem only applies to the case an APS channel is used.

A signal fail on protection must remove the external command FSw, if an APS channel is used. When the external request is denied it must be forgotten. *After the SF condition on protection has cleared, the external command was still active.*

This was specified in the text of the specification, but not correctly specified in the code. The incorrect behaviour is shown in Figure 6-8. In Figure 6-9 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command FSw for signal 1
6:   # |  0 |  0 | FSw ,1,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
8:   # |  1 |  1 | FSw ,1,1 >          |  0 |  1 |
9:   # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
10:  ### Node 1, command SF for signal 0
11:  # |  0 |  0 | SFL ,0,0 >          |  1 |  1 |
12:  # |  0 |  0 |          < RR  ,0,0 |  0 |  0 |
13:  ### Node 1, command NR for signal 0
14:  # |  1 |  1 | FSw ,1,1 >          |  0 |  0 |
15:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
16:  # |  1 |  1 | FSw ,1,1 >          |  1 |  1 |
```

**Figure 6-8 Forced switch is not removed at SF on protection: Incorrect behaviour**

```
Original code (in external request process)

1:   wait until CTimer is expired
2:   then   if (ERT==FSw) and not[((GRT=ERT) or (GRT=RR)) and (GRSN==ERSN)]
3:          then   ERT=NR
4:                 ERSN=0
5:          fi
6:          if ..
7:          fi
8:          ..
9:   tiaw

Corrected code

1:   wait until CTimer is expired
2:   while (ERT!=NR)
3:   do     if not[((GRT=ERT) or (GRT=RR)) and (GRSN==ERSN)]
4:          then   ERT=NR
5:                 ERSN=0
6:          fi
7:   od
```

**Figure 6-9 Forced switch is not removed at SF on protection: Correction in code**

### 6.2.4 DNR is not cleared after RR

This problem only applies to the non-revertive mode.

When there is no request, the previous selection must be maintained. The process state must be NR if protection is selected and DNR if the working link is selected. A DNR is replaced when a remote request occurs and consequently a RR is issued. The DNR state should be forgotten, *but the DNR state is issued again when it is the highest priority.*

The incorrect behaviour is shown in Figure 6-10. In Figure 6-11 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SD for signal 1
6:   # |  0 |  1 | SDL ,1,0 >          |  0 |  1 |
7:   # |  0 |  1 |          < RR  ,1,1 |  0 |  1 |
8:   # |  1 |  1 | SDL ,1,1 >          |  0 |  1 |
9:   # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
10:  ### Node 1, command NR for signal 1
11:  # |  1 |  1 | DNR ,1,1 >          |  1 |  1 |
12:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
13:  ### Node 2, command SD for signal 0
14:  # |  1 |  1 |          < SDL ,0,1 |  0 |  1 |
15:  # |  0 |  1 | RR  ,0,0 >          |  0 |  1 |
16:  # |  0 |  1 |          < SDL ,0,0 |  0 |  1 |
17:  # |  0 |  1 | RR  ,0,0 >          |  0 |  1 |
18:  ### Node 2, command NR for signal 0
19:  # |  0 |  1 |          < NR  ,0,0 |  0 |  1 |
20:  # |  0 |  1 | DNR ,1,0 >          |  0 |  1 |
21:  # |  0 |  1 |          < RR  ,1,1 |  0 |  1 |
22:  # |  1 |  1 | DNR ,1,1 >          |  0 |  1 |
23:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
```

**Figure 6-10 DNR is not cleared after RR: Incorrect behaviour**

```
Original code (in local request priority process)

1:   if (LRTnew==NR)
2:   then   if (OPERtype==non-revertive)
3:          then   if (LRSN==1)
4:                 then   LRT=DNR
5:                 else   LRT=NR
6:                 fi
7:          else ..


Corrected code

1:   if (LRTnew==NR)
2:   then   if (OPERtype==non-revertive)
3:          then   if (GRSN==1)
4:                 then   LRT=DNR
5:                 else   LRT=NR
6:                        LRSN=0
7:                 fi
8:          else ..
```

**Figure 6-11 DNR is not cleared after RR: Correction in code**

### 6.2.5 SF on protection does not remove SF for a working link

This problem only applies to the case an APS channel is used.

It is specified that a signal request does not remove a current signal request if it has the same priority. Although not specified, *a signal fail on protection must remove another signal fail request.*

The incorrect behaviour is shown in Figure 6-12. In Figure 6-13 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SF for signal 1
6:   # |  0 |  0 | SFL ,1,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
8:   # |  1 |  1 | SFL ,1,1 >          |  0 |  1 |
9:   # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
10:  ### Node 1, command SF for signal 0
11:  # |  0 |  0 | SFL ,1,0 >          |  1 |  1 |
12:  # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
```

**Figure 6-12 SF on protection does not remove SF for a working link: Incorrect behaviour**

```
Original code (in local request priority process)

1:   if (LRTnew==NR)
2:          ..
3:   else   if (LRsource==external)
4:          then   LRT=LRTnew
5:                 LRSN=LRSNnew
6:          else   if (LRTnew!=LRT)
7:                 then   LRT=LRTnew
8:                        LRSN=LRSNnew
9:                 else ..
10:  fi

Corrected code

1:   if (LRTnew==NR)
2:          ..
3:   else   if (LRsource==external)
4:          then   LRT=LRTnew
5:                 LRSN=LRSNnew
6:          else   if (LRTnew!=LRT) or (SRT/0==SF and APSmode==true)
7:                 then   LRT=LRTnew
8:                        LRSN=LRSNnew
9:                 else ..
10:  fi
```

**Figure 6-13 SF on protection does not remove SF for a working link: Correction in code**

### 6.2.6 Absent code for dropping WTR

It is specified that the process state WTR should be dropped and the associated timer cleared, when any request occurs. *This is not specified in the code.*

The incorrect behaviour is shown in Figure 6-14. In Figure 6-15 a correction in the source is presented.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SD for signal 1
6:   # |  0 |  0 | SDL ,1,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
8:   # |  1 |  1 | SDL ,1,1 >          |  0 |  1 |
9:   # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
10:  ### Node 1, command NR for signal 1
11:  # |  1 |  1 | WTR ,1,1 >          |  1 |  1 |
12:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
13:  ### Node 2, command SD for signal 2
14:  # |  1 |  1 |          < SDL ,2,1 |  0 |  1 |
15:  # |  0 |  2 | RR  ,2,2 >          |  0 |  1 |
16:  # |  0 |  2 |          < SDL ,2,2 |  2 |  2 |
17:  # |  2 |  2 | RR  ,2,2 >          |  2 |  2 |
18:  ### Node 2, command NR for signal 2
19:  # |  2 |  2 |          < WTR ,2,2 |  2 |  2 |
20:  # |  2 |  2 | RR  ,2,2 >          |  2 |  2 |
21:  ### Node 2, command TO
22:  # |  2 |  2 |          < NR  ,0,2 |  0 |  2 |
23:  # |  0 |  0 | WTR ,1,0 >          |  0 |  2 |
24:  # |  0 |  0 |          < RR  ,1,1 |  0 |  1 |
25:  # |  1 |  1 | WTR ,1,1 >          |  0 |  1 |
26:  # |  1 |  1 |          < RR  ,1,1 |  1 |  1 |
```

**Figure 6-14 Absent code for dropping WTR: Incorrect behaviour**

```
Added code (after original code in local request priority process)

1:   while (LRT==WTR) or (LRT==DNR)
2:   do
3:          if (LRT==WTR)
4:          then    if (WTRtimer==0) or (GRT!=WTR)
5:                  then    LRT=NR
6:                          if (EXTRAtraffic==true)
7:                          then    LRSN=Nmax+1
8:                          else    LRSN=0
9:                          fi
10:                 fi
11:         fi
12:         if (LRT==DNR) and (GRSN!=1)
13:         then    LRT=NR
14:                 LRSN=0
15:         fi
16:  od
```

**Figure 6-15 Absent code for dropping WTR: Correction in code**

## 6.3 Open issues

Some observations of the behaviour of the ETSI specification could not fully be investigated within the graduation project. They are listed in this paragraph and possible adjustments to the source is proposed.

### 6.3.1 Selector is released at SF on protection

This item only applies to the 1:N unidirectional case.

If protection fails in one direction, then protection in this direction is of no use and the corresponding selector and bridge must release. Protection in the opposite direction is still possible. But as also the APS channel is broken in the failing direction, no bridge control is possible and the corresponding selector and bridge (opposite direction) should *freeze*.

A part of this was specified in the G.783 standard and adopted straight in the ETSI standard: the freezing of the bridge was specified, but the specification of the freezing of the selector is absent.

When one protection link is already failing, the question is: what must happen when also the other protection fails. The behaviour as specified in the ETSI specification is shown in Figure 6-16.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   ### Node 1, command SD for signal 1
6:   # |  0 |  0 | SDL ,1,0 >          |  0 |  0 |
7:   # |  0 |  0 |          < NR  ,0,1 |  0 |  1 |
8:   # |  1 |  0 | SDL ,1,0 >          |  0 |  1 |
9:   ### Node 2, command SF for signal 0
10:  # |  1 |  0 |          < SFL ,0,1 |  0 |  1 |
11:  # |  1 |  0 | SDL ,1,0 >          |  0 |  1 |
12:  ### Node 1, command SF for signal 2
13:  # |  0 |  0 | SFL ,2,0 >          |  0 |  1 |
14:  # |  0 |  0 |          < SFL ,0,1 |  0 |  1 |
```

**Figure 6-16 Selector is released at SF on protection**

### 6.3.2 Extra traffic is removed on SD on protection

Both in the G.783 standard and in the ETSI standard, the operation concerning extra traffic is not specified completely. It is clear that extra traffic must be removed, if any request occurs that needs the protection link. It is also clear that extra traffic has not much use when the protection has a SF condition. *But there is nothing specified for a SD on protection.*

If the highest request is SD on protection, extra traffic can still be transported over the degraded links.

The behaviour is shown in Figure 6-17 for N=2, so the extra traffic signal has number 3.

```
1:   # 1:N, bidir, revertive, APS=1, EXTRAtraffic=0
2:   # |----|----|----------|----------|----|----|
3:   # |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|
4:   # |----|----|----------|----------|----|----|
5:   # |  0 |  0 | NR  ,3,0 >          |  0 |  0 |
6:   # |  0 |  0 |          < NR  ,3,3 |  0 |  3 |
7:   # |  3 |  3 | NR  ,3,3 >          |  0 |  3 |
8:   # |  3 |  3 |          < NR  ,3,3 |  3 |  3 |
9:   ##|----|----|----[initialized]----|----|----|
10:  ### Node 1, command SD for signal 0
11:  # |  0 |  3 | SDL ,0,3 >          |  3 |  3 |
12:  # |  0 |  3 |          < RR  ,0,0 |  0 |  0 |
13:  # |  0 |  0 | SDL ,0,0 >          |  0 |  0 |
14:  # |  0 |  0 |          < RR  ,0,0 |  0 |  0 |
```

**Figure 6-17 Extra traffic is removed on SD on protection**

### 6.3.3 Alternative structure to distinct normal and SF/0 case better

As mentioned in paragraph 4.4, the behaviour of the APS protocol can be described independently for the normal case and for the case there is a SF/0 condition. This separation is not made in the ETSI specifications.

Several suggestions can be made:
- for each process separate code can be written that addresses SF/0 behaviour,
- new internal states can provide a distinction within the current structure,
- perhaps some choices with respect to the SF/0 behaviour can be made that makes it more simple.

A method to determine that a newly written structure for the specifications has identical behaviour to the original specification is the *conformance test*. This is a verification in which all possible stimuli are supplied and the output behaviour is checked for identity between the two specifications.

### 6.3.4 Handling of an all-ones APS message

When a link has a fail condition in a SDH network, the bits in the frames are filled with ones. This also affects the APS fields in the APS channel. Such a all-ones APS messages could be received before the protection process notices the failing condition of the link and a not intended APS message could be received. What such a message exactly is and what behaviour the protection process consequently has is for further investigation.

# 7 Graphical Simulations

While performing simulations using SPIN, some shortcomings and desires have arisen. One of them is the fact that all simulations were textual. In this chapter these observations are discussed (7.1) and a tool that improves some simulation aspects is presented in 7.2. In 7.3 the interface, commands and options of an actual implementation of this tool is presented. Finally a structure description is discussed in 7.4

## 7.1 Simulation Data Presentation

It is important how simulation data is presented to the designer. If simulation data is hard to interpret, then the behaviour of the system could be unclear or the overview could be lost when looking at details.
The designer uses the simulation information to get an overview of the system behaviour, to get insight in behaviour of a specific part of the system or to get details of the behaviour.

Another important aspect of graphical simulations is that the behaviour of a system can easily be shown the another technical expert or to a generalist, who wants an overview and is interested in details.

The output of simulations with the model checker SPIN is textual. This textual output restricts the simulation data to words, values and a limited form of sketching out behaviour, e.g. the direction messages are flowing (see Figure 6-1). This was adequate for simulations of the APS protocol, but could be inadequate for network simulations with for example ten nodes.

With only the possibility to produce textual output, there must be thought of ways to cope with the data abundance. Here are some ways:
- the information can be made more compact by writing short cryptic codes,
- the information can be made more orderly by means of structuring,
- only the information of a specific aspect of the system can be output,
- from the information, a specific aspect can be picked out by means of data post-processing.

But as simulations is concerned, a observation can be made. The designer always has a certain image of the system he is designing. If the designer is simulating a network protocol, this image could be a graph of nodes and links with certain notations of messages that flow across the network. This image is not necessarily the same as the form of elements in the simulation model.
When interpreting simulation data, the designer translates this data into his image of the system. The translating could (partially) also be done by the computer.

In order to make the computer able to make this translation, the designer must tell it how his image of the system looks. In other words, the designer makes a decision how he wants to visualize the system. When this visual model is finished, the designer tells the computer how the simulation data is presented in this visualization.

Fundamental is the uncoupling of the structure of the system model from the structure of the simulation model. Although the structure of the system model can be complex, the structure of the simulation can be tailor-made to the desired detail level.

An example of a situation where graphical simulation is desirable is shown in Figure 7-1.

> In this example the behaviour of linear protection switching in a group of three SNC protection rings is tested. A SNC protection ring is a linear protection topology on top of a ring network. In this case, dual node interconnection is used. This means that the rings are connected with two links. This is a method to increase the availability of the interconnection of rings.
>
> Note that SNC ring protection is 1+1 unidirectional. The selecting node (marked with a 'X' in the figure) chooses the signal from either the east or west side. The assignment the predicate working and protection is not very useful here, except to assign a default selection. Both nodes in the origin of the dual node interconnection are supplied with the east and west signal.
>
> In this configuration the hold-off problems, discussed in paragraph 4.3, have a clear application (partitioned protection) and can be the very subject of the simulation.
> The model of this system in PROMELA can contain multiple processes and channels per node and can have complex behaviour considering the node internal. This is hidden in the graphical simulation.
> Textual output of SPIN simulation output could be very disorderly. Every relevant state of the nodes and links would be denoted with a node/link identity number and data considering its state. A graphical presentation is clearly a great improvement of insight in behaviour.

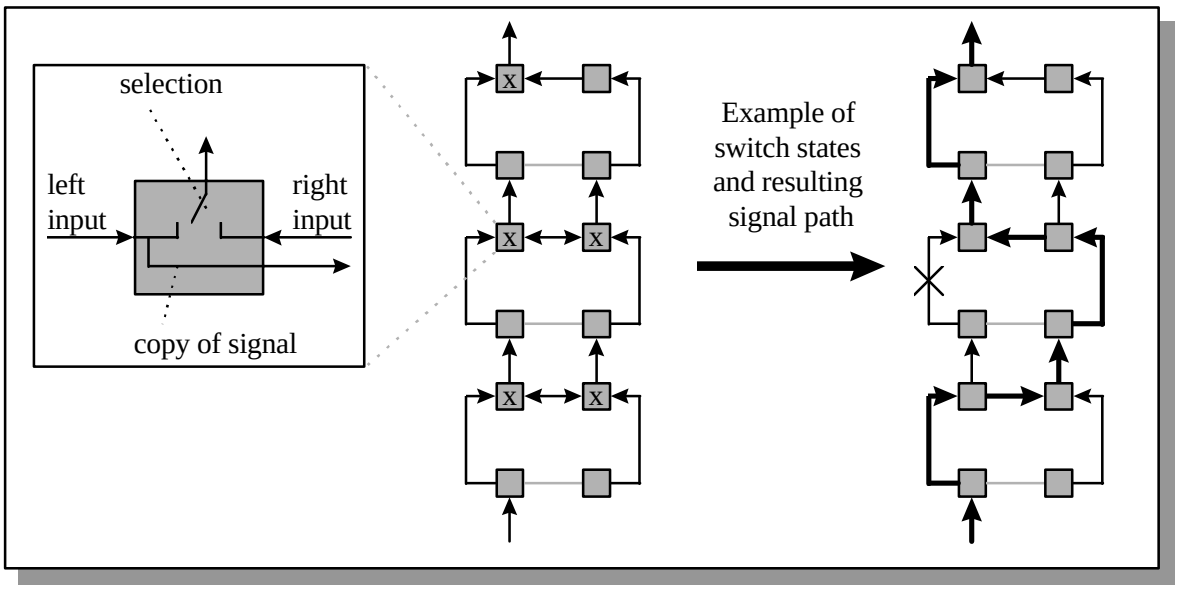



**Figure 7-1 Three SNC protection rings with dual node interconnection**

## 7.2 A Graphical Simulator: InSPIN

From the mentioned needs, a graphical simulation environment is developed. The objective is a make simulations graphical, not to reinvent the wheel. The tool SPIN is excellent for model simulations and the idea of the intended environment is to develop a graphical shell that uses SPIN for the simulation and modelling and adds extra features to meet the developers needs. This shell is called *InSPIN* [1]. The essence of this approach is that the same PROMELA model can be used for simulations and validation with SPIN as for graphical simulations and presentations with InSPIN.

As mentioned in paragraphs 5.4 and 6.1, SPIN uses a text file with the PROMELA model as input and produces lines of text as output. To interface with SPIN, InSPIN must provide input and read the output of SPIN. SPIN is a console program that accepts several parameters at the command prompt. One parameter specifies the file, which is the PROMELA model. The output is delivered on the console. This output van also be redirected the file. InSPIN uses this mechanism to communicate with SPIN. See Figure 7-3.

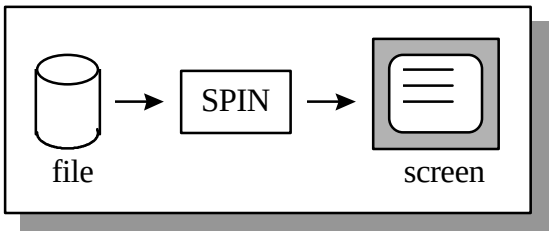



**Figure 7-2 SPIN in- and output**

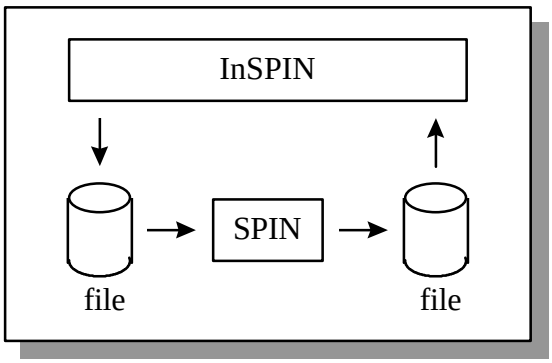



**Figure 7-3 Communication with SPIN**

In the PROMELA model, certain expressions could be printed to describe the state of an object. InSPIN interprets all SPIN output and reacts in a predefined manner to these expressions.

Before designing InSPIN, several design objectives were formulated that are useful for the designer. They are presented here as the InSPIN feature groups.

[1] Interface for SPIN

**Project features**
- Grouping of all information in a project.
- A project consists of the PROMELA model, the graphical layout, the simulation results and possible comments.
- It is possible present simulation results at a later time, enabling the designer to demonstrate his model.

**Simulation presentation features**
- Possibility to present any value in model in a graphical or textual manner.
- Graphical presentation consist of blocks and arrows. The properties of these graphical objects change their appearance and are affected by the simulation.
- Textual presentation consists of infoboxes, placeable anywhere on the screen. They display text that can be dictated by the simulation.
- Stepping through simulation results.
- This stepping can be related to time (if modeled) or to stimulus events.

**Simulation design features**
- Possibility to assign variables in the SPIN model from the InSPIN environment.
- Specifying PROMELA statements as simulation stimuli in the simulation environment.
- The generation of PROMELA statements as stimuli from user indicated graphical events (e.g. the designer clicks on an arrow to indicate the failure of a link).

**Layout design features**
- Possibility to design a layout with elements specified in the PROMELA model and to convert the layout into the correct PROMELA code. This covers the instantiation of processes and declaration of channels.

The project feature group is easily implemented and of high practical use. The simulation presentation feature group is the essence of this chapter and is an absolute goal. The simulation and layout design features groups have the objective to take more routine work out of the designers hands.
The last two feature groups are harder to implement and do not have such a direct practical use as the first two, because they rely on the PROMELA language and on the modelling style. The InSPIN version that is realized in this graduation project, only attempts to satisfy the project and simulation presentation feature groups.

## 7.3 InSPIN interface and commands

InSPIN features four graphical elements that form a minimal set to enable network simulations. They are presented in Figure 7-4 and listed with their properties in Table 7-1. Properties of elements determine its appearance. Properties that can only be appointed at the design of the layout are called static. Dynamic properties can be altered with commands.

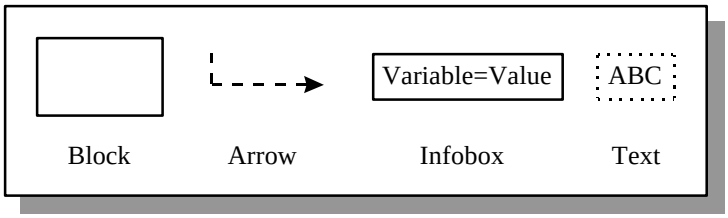



**Figure 7-4 The InSPIN graphical elements**

**Table 7-1 The InSPIN graphical elements**

| **Element** | **Static properties** | | **Dynamic properties** | |
|---|---|---|---|---|
| Block | Name<br>Shape | (string)<br>(rectangle, round rectangle, ellipse) | View | (string, indicating an image) |
| Arrow | Name | (string) | Drawstyle<br>Direction | (thin, thick, dashed)<br>(none, up, down, both) |
| Infobox | Name<br>Variable | (string)<br>(string) | Value | (string) |
| Text | Name<br>Text | (string)<br>(string) | - | |

The dynamic property *view* of the block element needs an explanation. During layout design, images can be supplied and associated with a name, called view. When a view is assigned to the view property, the corresponding image is drawn inside the block element. This provides a basic means to visualize an internal structure of a block. In the protection switching simulations, this is used to visualize the state of the bridge and selector.

The assignment of a view is only possible when the shape property is rectangle. Also the images must be rectangle.

With use of the `printf` statement, commands are send from the PROMELA model to the InSPIN layout. A command is a piece of text that affects a property of a graphical object. It has the following structure:

```
#elementname.property=value#
```

For example, a command like '`#link0.drawstyle=dotted#`' changes link0 into a dotted line.

The InSPIN interface is shown in Figure 7-5, with an example of a linear protection switching model.

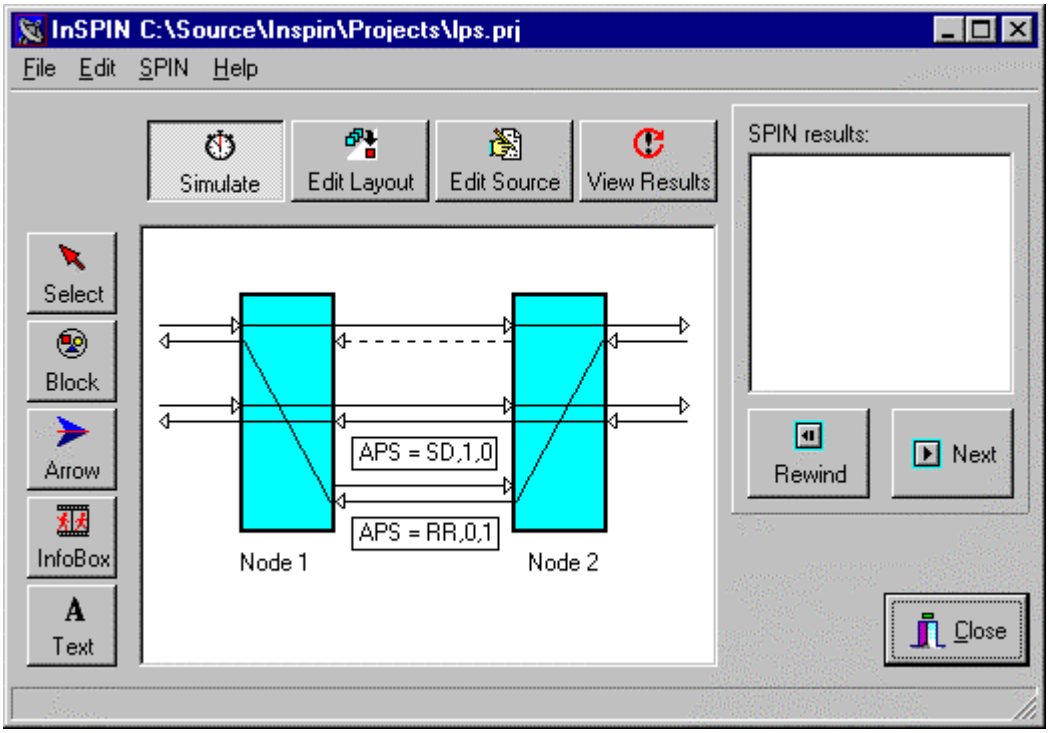



**Figure 7-5 The InSPIN interface**

## 7.4 The Structure of InSPIN

InSPIN is implemented in Delphi[1]. The language of Delphi is Object Pascal, a object oriented extension to PASCAL. It has many similarities with C++. Here an overview is presented of the general structure.

First some terms of Object Pascal are explained, which are needed in the discussion of the structure:

**Object** An object is an instance of a class. It can be created and destroyed dynamically.

**Class** A class is a data type. It encapsulates related data and code for operating on that data, known as methods. The data and methods are collectively known as class members. Classes can inherit members of other classes.

**Inheritance** A class that is derived from another class, inherits its class members. It is said to be a descendant of that class.

**Ownership** A objects is owner of another object if it has instantiated it. It also has the responsibility to destroy it.

**Reference** A reference to an object is the memory address of the instantiation of that object. It is the only handle that enables access to its members. Besides the owner, other objects can also store the reference to an object. A least the owner must store the reference to enable destruction.

**Object structure**

[1] Delphi a Windows development tool from Borland International. Windows is a graphical operating system from Microsoft Corporation. Both run in a PC environment.

In Figure 7-6 the class inheritance structure of InSPIN is shown. The classes printed in bold are instantiated in the object structure, the normal printed classes are used to specify data and code that is common to their descendant classes. The class ‘TObject’ is the root class of all classes in Delphi. It basically provides the means to enable creation and destruction of objects.

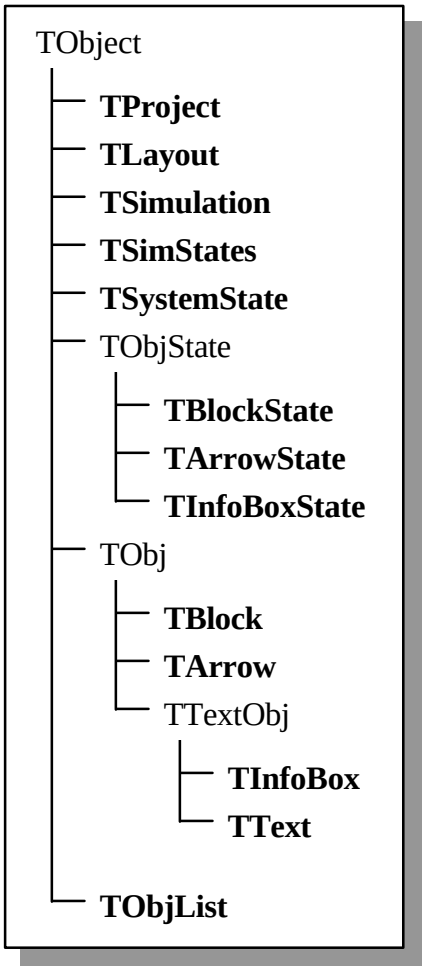



**Figure 7-6 InSPIN class inheritance structure**

A short description of the class definition follows:

The classes *TBlock*, *TArrow*, *TInfoBox* and *TText* implement the graphical elements of InSPIN. They determine the look of the object in the layout. Their predecessor class *TObj* provides data and code to name the individual object and to enable listing them in the *TObjList* class.

The descendants of the *TObjState* class handles the dynamic properties of the graphical elements. Each instantiation is associated with a TObj instantiation. It represents a state of an TObj object. The *TSystemState* class maintains a list of all TObjState states. It represents the total system state. The *TSimStates* class maintains a list of TSystemState objects. It represents all steps in a simulation.

The *TSimulation* class represents a simulation. It associates a PROMELA source file with a TSimStates object to represent the states of a simulation.

The *TLayout* class represents a layout design. It owns four lists of the TObjList class, one for each graphical element object, and a TSystemState object representing the initial state of the model.

The *TProject* class handles the total simulation project. It owns a TLayout object and a list of TSimulation objects that each represent a performed simulation.

In Figure 7-7 the object structure of InSPIN is shown.

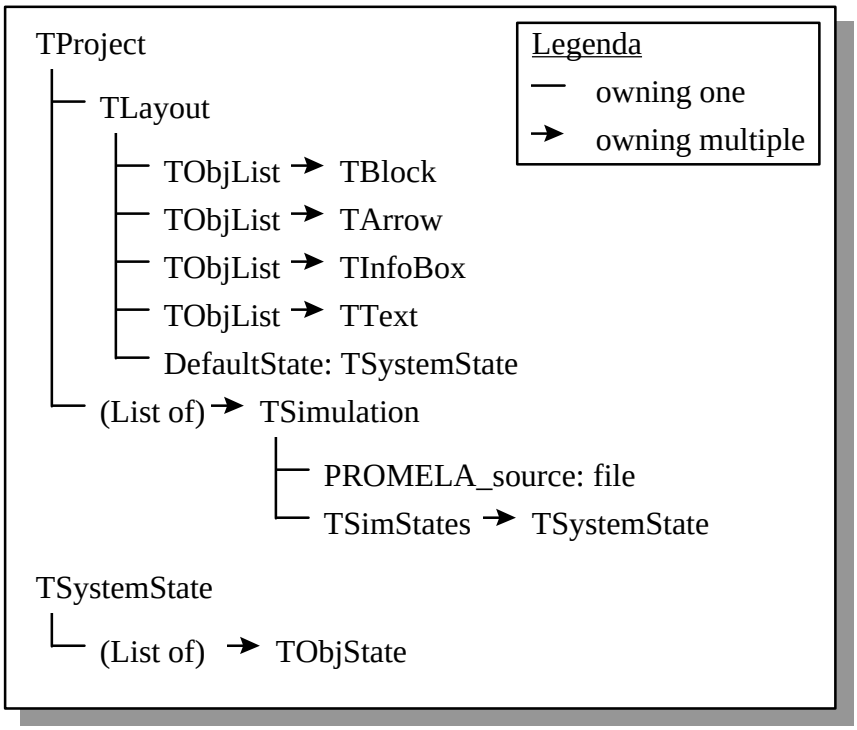



**Figure 7-7 InSPIN object structure**

### Layout design

Initially the project contains an empty layout and an empty list of simulations. The layout contains four empty list of graphical element objects. Each graphical element the designer creates is inserted in the corresponding list in the layout.

The element position and static properties are stored in the element object. The dynamic properties that are assigned to the element are stored in the elements currentstate object, which is a corresponding descendant of TObj State. All currentstate objects of the graphical elements are also stored in a defaultstate object of type TSystemState in the layout.

### PROMELA model editing

When the designer enters the source editing mode, a text editor window is shown in which the designer can edit the PROMELA code. It is saved in a file and this filename is stored in a simulation object of type TSimulation.

**SPIN simulation execution**

When the designer runs the simulation, InSPIN uses the entered PROMELA file as input for SPIN. The output that SPIN produces is captured and presented in the results window. InSPIN also uses this result text to build a simulation state list of type TSimStates. The first entry in the SimState list is the defaultstate of the layout. Every command that is found in the output text generates a new systemstate in the list. Although a command only affects one graphical element, the complete system state is calculated and stored. This makes stepping back through the simulation more easy.

**Simulation stepping**

When the simulation is completed, the designer can step through the simulation states. Every entry that is indicated by the designer, is used to draw up the layout. The system behaviour can be observed.

# 8 Conclusions

**General**

System specifications should be written as formal and unambiguous as possible to enable verification. In order to describe systems, a language is needed and in order to verify a model, a verification tool is needed. Performing the verification on a computer is absolutely necessary, because even simple communication systems or protocols can be to large and complicated to verify manually or mathematically.

**Modelling and verification**

The specification and verification of systems demand several properties of the specification language and the verification tool. Several languages and tools are tested for these properties as described in paragraph 5.1.

From this investigation, one tool and its related language is found to be best suitable for the verification of the linear protection switching specification. This tool is called SPIN and its modelling language is called PROMELA. This tool is easy to use and has a large validation power, which is required for large and complex systems. The language is intuitive, easy to learn and has enough expression power to model the system specifics related to protection switching behaviour. Also, SPIN has a wider application area; most systems found in telecommunication and information technology today are can well be modelled and verified with SPIN.

**Verification results**

This tool is used to verify the ETSI linear protection switching specification. Several flaws were found. They are described in paragraph 6.2. Also proposals are presented that correct these flaws. In paragraph 6.3, some findings are listed that need future attention. The findings and an initial solution are described.

**Shortcomings of SPIN**

While simulating with the SPIN tool, it was noticed that the simulation output is somewhat limited. SPIN only produces textual output. To enable graphical simulations of systems of any appearance, a tool called InSPIN is developed. This tool relies on SPIN and its language PROMELA for the system description and translates the simulation results in a user-defined graphical manner. This graphical simulator appeared adequate in simulating linear protection switching operation.

**Follow up work**

The hold-off problems (discussed in paragraph 4.3) needs future attention.

More research is still to be done in the different system aspects and the modelling techniques they require. Also more verification tools and languages must be investigated to come to a complementary set of verification techniques.

Further detailed development work of the InSPIN tool is recommended to make the tool features more accessible to other potential users.

System specification, modelling and verification is an area in which a lot of research and development of languages and tools is needed. A lot of opportunities remain for graduation and Ph.D. students for further research and development in this area of system architecture and requirements.

# References

[Arn94] J.C. Arnbak, *Collegedictaat Transmissiesysteemtechniek*, Vakgroep Telecommunicatie- en Verkeersbegeleidingssystemen (TVS), 1994.

[Hol91] C.J. den Hollander, *SDH Fundamentals*, Trends in Telecommunications, Volume 7, no.2, 1991.

[Holz91] G.J. Holzmann, *Design and Validation of Computer Protocols*, Englewood Cliffs, N.J.: Prentice Hall, ISBN 0-13-539925-4, 1991.

[NW96] M. Nagel and A. Willems, *Modelling and Simulation of SDH-Networks Summary and Proposals*, JNL216-R-96005, Lucent Technologies, 1996.

[PV97] P.H.A. van der Putten and J.P.M. Voeten, *Specification of Reactive Hardware/Software Systems*, Ph.D. thesis, Technical University Eindhoven, Eindhoven, The Netherlands, 1997.

[SR91] M. Sexton and A. Reid, *Transmission Networking: SONET and the Synchronous Digital Hierarchy*, Norwoord, MA: Artech House, ISBN 0-89006-551-9, 1992.

[Wal91] J. Walrand, *Communication Networks: A First Course*, Homewood, IL: IRWIN, ISBN 0-256-08864-0, 1991.

# Annex A

## Generic Specification of Linear Protection Switching operation (ETSI)

# DE-TM-01015-3-1

**FINAL DRAFT**
pr**ETS 300 417-3-1**

**EUROPEAN**
**TELECOMMUNICATION**
**STANDARD**

**Version: First Edition**

**for TM approval**

**Date: September 1996**

Source: ETSI TC-TM

Reference: DE-TM-01015-3-1

ICS: 33.020

**Key words:** transmission, SDH, interface

# Transmission and Multiplexing (TM); Generic requirements of transport functionality of equipment; Part 3-1: STM-N regenerator and multiplex section layer functions

**ETSI**

European Telecommunications Standards Institute

**ETSI Secretariat**

**Postal address:** F-06921 Sophia Antipolis CEDEX - FRANCE
**Office address:** 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE
**X.400:** c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 92 94 42 00 - Fax: +33 93 65 47 16

Whilst every care has been taken in the preparation and publication of this document, errors in content, typographical or otherwise, may occur. If you have comments concerning its accuracy, please write to "ETSI Editing and Committee Support Dept." at the address shown on the title page.

# Annex A (normative): Generic specification of linear protection switching operation

NOTE 1: The text in this annex is a reworked copy of Annex A of ITU-T Recommendation G.783 [5] and presents an attempt to formalise the protection process specification to remove ambiguities present in ITU-T Recommendation G.783 [5].

The protection process described in this annex supports linear trail protection (ETS 300 417-1-1 [1], subclause 9.3.1) as well as linear connection (subnetwork, network) protection (ETS 300 417-1-1 [1], subclauses 9.4.1 and 9.4.2) in the combinations as listed in table A.1. This protection process controls the bridge and selector functionality (ETS 300 417-1-1 [1], subclause 9.2, figures 47, 49).

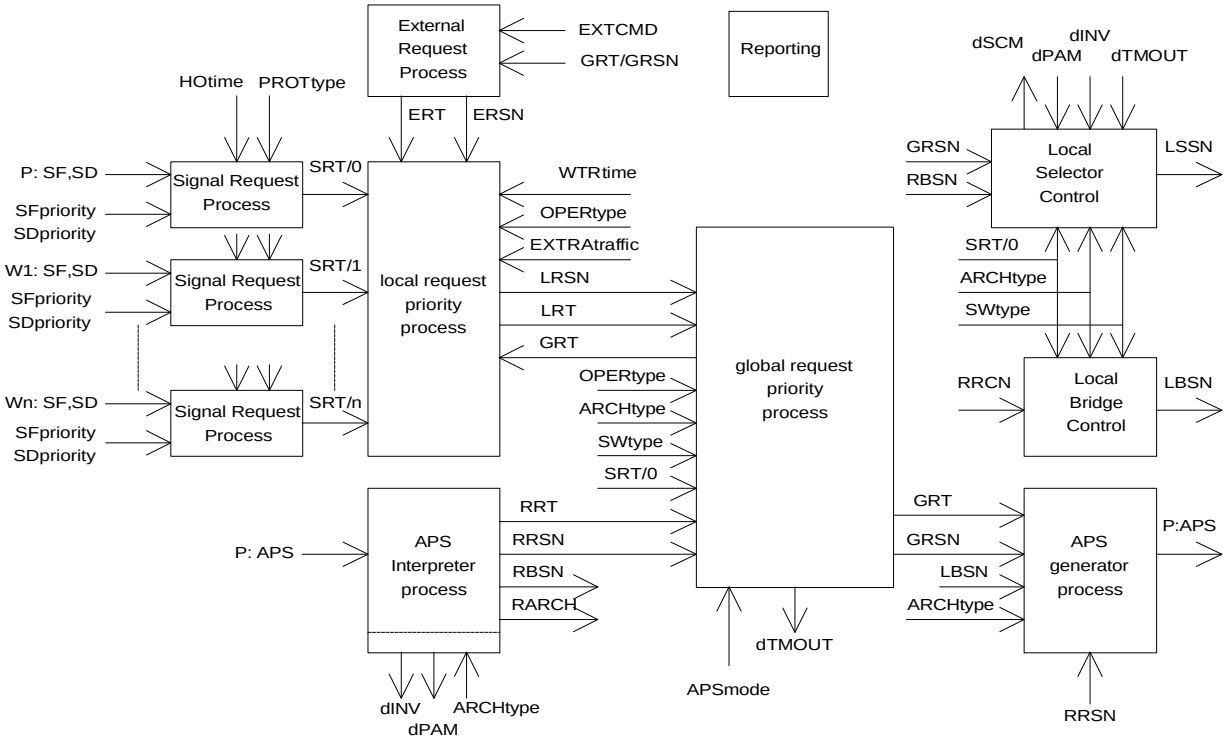
**Table A.1: Supported linear protection process combinations**

| Protection type | Architecture type | Switching type | Operation type | APS signal | Extra traffic |
|---|---|---|---|---|---|
| MS-n trail | 1+1 | uni-directional | non-revertive | no (note) | no |
| MS-n trail | 1+1 | uni-directional | revertive | no (note) | no |
| MS-n trail | 1+1 | bi-directional | non-revertive | yes | no |
| MS-n trail | 1+1 | bi-directional | revertive | yes | no |
| MS-n trail | 1:n ($n \leq 14$) | uni-directional | revertive | yes | no |
| MS-n trail | 1:n ($n \leq 14$) | bi-directional | revertive | yes | no |
| MS-n trail | 1:n ($n \leq 14$) | bi-directional | revertive | yes | yes |
| VC-m SNC/I | 1+1 | uni-directional | non-revertive | no | no |
| VC-m SNC/I | 1+1 | uni-directional | revertive | no | no |
| VC-m SNC/N | 1+1 | uni-directional | non-revertive | no | no |
| VC-m SNC/N | 1+1 | uni-directional | revertive | no | no |
| VC-m SNC/S | 1+1 | uni-directional | non-revertive | no | no |
| VC-m SNC/S | 1+1 | uni-directional | revertive | no | no |
| VC-m trail | 1+1 | uni-directional | non-revertive | no | no |
| VC-m trail | 1+1 | uni-directional | revertive | no | no |
| VC-m trail | 1+1 | bi-directional | non-revertive | yes | no |
| VC-m trail | 1+1 | bi-directional | revertive | yes | no |
| note: G.783 specifies the use of the APS in these cases, without taking any action on the information. | | | | | |

NOTE 2: Bi-directional switched 1+1 VC-m trail protection requires the definition and bit allocation of the VC-APS signal.

The remainder of this annex is organised as follows:
- protection process overview;
- external commands definition;
- conditions of protected trail/connection signals;
- states within protection process;
- numbering of working, protection, normal, extra traffic and null signals;
- numbering and priority of external commands, trail/connection signal conditions, and states;
- automatic protection switch (APS) signal definition;
- specification of subprocesses within protection process.

## A.1 Protection process overview




| | |
|---|---|
| LRT: | Local Request Type |
| LRSN: | Local Request Signal Number |
| LBSN: | Local Bridge Signal Number |
| LSSN: | Local Selector Signal Number |
| RRT: | Remote Request Type |
| RRSN: | Remote Request Signal Number |
| RBSN: | Remote Bridge Signal Number |
| RARCH: | Remote Architecture Type |

| | |
|---|---|
| ERT: | External Request Type |
| ERSN: | External Request Signal Number |
| GRT: | Global Request Type |
| GRSN: | Global Request Signal Number |
| SRT: | Signal Request Type |
| SRSN: | Signal Request Signal Number |

**Figure A.1: Subprocesses within generic linear trail/connection protection processes**

Linear protection processes can be characterised by the following (super)set of subprocesses (figure A.1):

| | |
|---|---|
| Signal Request | converts SF and SD signals of a working/protection trail/connection signal into a (signal) request type and trail/connection number |
| External Request | converts the external commands into an (external) request type and signal number |
| Local Request Priority | determines the highest priority local request |
| APS Interpretation | converts the APS signal into a (remote) request type, request signal number, bridged signal number, and architecture type (if applicable) |
| Global Request Priority | determines the highest global request type comparing local and remote (if applicable) requests |
| Local Bridge Control | determines which of the normal/extra traffic signals is bridged to the protection trail/connection |
| Local Selector Control | determines which of the normal/extra traffic signals is connected to/extracted from the protection trail/connection |
| APS Generation | converts the global request type, global request signal number, local bridged signal number, and local architecture into the APS signal |
| Reporting | reports the status (local, remote) of the protection process; remote status if APS signal is supported |

A specific protection application is characterised by the following parameter set:

**Table A.2**

| Parameter | Value options |
|---|---|
| Architecture type (ARCHtype) | 1 + 1, 1:n |
| Switching type (SWtype) | uni-directional, bi-directional |
| Operation type (OPERtype) | revertive, non-revertive |
| APS signal (APSmode) | true, false |
| Wait-To-Restore time (WTRtime) | in the order of 0-12 minutes |
| Switching time | ≤50 ms |
| Hold-off time (HOtime) | 0 to 10 seconds in steps of the order of 100 ms |
| Protection type (PROTtype) | SNC/I, SNC/N, SNC/S, trail |
| Signal switch conditions: | SF = SSF (SNC/I)<br>SF = TSF (SNC/N, SNC/S, trail), SD = TSD (SNC/N, SNC/S, trail) |
| External commands (EXTCMD) | LO-#0, FSw-#i, MSw-#i, EXER-#i, CLR |
| Extra traffic (EXTRAtraffic) | true, false |

## A.2 External switch commands definition

A switch command issues an appropriate external request. Only one switch request can be issued per protection group. Switch commands are listed below in the descending order of priority and the functionality of each is described.

NOTE 1: The addition of the Lockout for Working #i command is for further study.

The function shall generate an automatic response confirming that the request was executed, or stating that the request was denied for a particular reason.

1) Clear (CLR): Clears all switch commands listed below.

2) Lockout of protection (LO): Request to deny all normal signals (and the extra traffic signal, if applicable) access to the protection trail/connection.

   NOTE 2: Request is honoured unless an equal priority switch command is in effect. If the request is denied, it is released and forgotten.

3) Forced switch #i (FSw-#i): Request to switch normal signal #i ($1 \leq i \leq n$, $n \leq n_{max}$) to the protection trail/connection, or request to switch extra traffic signal #$n_{max}$+1 to the protection trail/connection, or (for the case of 1+1 non-revertive systems) request (FSw-#0) to switch normal signal to working trail/connection.

   NOTE 3: Request is honoured unless an equal or higher priority switch command is in effect or (for the case an APS signal is in use) SF condition exists on the protection trail/connection. If the request is denied, it is released and forgotten.

   NOTE 4: For 1 + 1 non-revertive systems, "forced switch no normal signal (FSw-#0)" transfers the normal signal from protection to the working trail/connection, unless an equal or higher priority request is in effect. Since forced switch has higher priority than SF or SD on the working trail/connection, this command will be carried out regardless of the condition of the working trail/connection.

   NOTE 5: For 1: n architectures, “forced switch to extra traffic (FSw-#$n_{max}$+1)” forces the extra traffic signal to the protection trail/connection and prevents normal signals to be transported over protection.

4) Manual switch #i (MSw-#i): Request to switch normal signal #i ($1 \leq i \leq n$, $n \leq n_{max}$) to the protection trail/connection, or (for the case of 1+1 non-revertive systems) request (MSw-#0) to switch normal signal to working trail/connection.

   NOTE 6: Request is honoured unless a defect condition exists on other trail/connections (including the protection trail/connection) or an equal or higher priority switch command is in effect. If the request is denied, it is released and forgotten.

   NOTE 7: For 1 + 1 non-revertive systems, "manual switch no normal signal (MSw-#0)" transfers the normal signal back from protection to the working trail/connection, unless an equal or higher priority request is in effect. Since manual switch has lower priority than SF or SD on a working trail/connection, this command will be carried out only if the working trail/connection is not in SF or SD condition.

5) Exercise #i (EXER-#i): Request for an exercise to check responses on APS bytes for normal signal #i ($1 \leq i \leq n$, $n \leq n_{max}$). The switch is not actually completed, i.e. the selector is released by an exercise request on either the sent or the received and acknowledged K1 byte.

   NOTE 8: Request is honoured unless the protection signal is in use.

The following table presents alternative user interface external command strings for the case of 1+1 protection architectures. Note that the generic names will be used in this ETS.

**Table A.3**

| generic | alternative for 1+1 revertive | alternative for 1+1 non-revertive | result |
|---|---|---|---|
| LO (#0) | LO | - | normal signal connected to working trail/connection |
| FSw-#0 | - | FSw-(to)-W | normal signal connected to working trail/connection |
| FSw-#1 | FSw-(to)-P | FSw-(to)-P | normal signal connected to protection trail/connection |
| MSw-#0 | - | MSw-(to)-W | normal signal connected to working trail/connection |
| MSw-#1 | MSw-(to)-P | MSw-(to)-P | normal signal connected to protection trail/connection |

## A.3 Conditions of working and protection trail/connections

Working and protection trail/connection (signals) have a condition associated with them: fault free, signal fail, signal degrade. The condition is communicated with the protection process by means of the SF and SD signals within the characteristic or adapted information of the working/protection trail/connection signal.

## A.4 States within protection process

The protection process has a number of so called states associated with it: no request, do not revert, reverse request, and wait to restore. A description of the effect of the states is presented below:

*Wait to restore (WTR):* In the revertive mode of operation, the normal signal will be restored (i.e. the signal on the protection trail/connection will be switched back to the working trail/connection) when the working trail/connection has recovered from the fault.

To prevent frequent operation of the selector due to an intermittent fault, a failed working trail/connection must become fault-free. After the failed trail/connection meets this criterion, (and no other externally initiated commands are present) a fixed period of time will elapse before it is used again by the normal signal. During this WTR state, switching will not occur.

An SF or SD condition will override the WTR. After the WTR period is completed, a No Request state will be entered. Switching will then occur from the protection trail/connection to the working trail/connection.

*Reverse request:* For the case of bi-directional switching, a reverse request is returned for exerciser and all other requests of higher priority. This clearly identifies which end originated the switch request.

If the head end had also originated an identical request (not yet confirmed by a reverse request) for the same signal, then both ends would continue transmitting (in the APS signals) the identical request type (RT) and signal number (RSN) and perform the requested switch action.

In uni-directional switching, reverse request is never indicated.

Both wait-to-restore and do not revert requests in the RT fields of the transmitted APS signal are normally acknowledged by a reverse request in the RT field of the received APS signal. However, no request is acknowledged by another no request received.

*Do not revert:* In the non-revertive mode of operation, assuming the normal signal is on protection when the working trail/connection is repaired or a switch command is released, the tail end maintains the selection and issues LRT/LRSN = DNR/1 (do not revert for normal signal 1).

For the case of bi-directional switching, the head end also maintains the selection and continues indicating reverse request. The do not revert is removed when pre-empted by a defect condition or an external request.

*No request:* This state represents the inactive state of the request processes (signal, external, local, remote, and global request processes). None of the trail/connection signal conditions is active, none of the external commands is active, and none of the states described above is active.

## A.5 Numbering of working, protection, normal, extra traffic, null signals

The protection trail/connection shall be referred to as number "0". The working trails/connections are numbered "1", "2", etc. The assignment of these numbers to physical entities in a network element is equipment specific and not within the scope of this ETS.

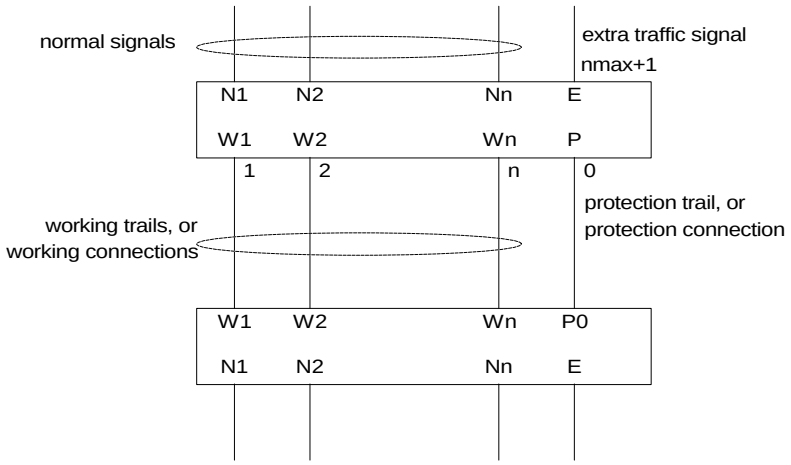



**Figure A.2: Definitions of working trail/connection, protection trail connection, normal and extra traffic signal**

The normal signals shall be numbered (equivalent to the working trails/connections) "1", "2", etc. In 1:n ($n = 1,2,3,..,n_{max}$) protection architectures normal signal #i shall be transported over working trail/connection #i or over the protection trail/connection. For the case of section layer protection, the assignment of these numbers to physical entities in a network element is equipment specific and not within the scope of this ETS. For the case of path layer protection, the assignment of these numbers to physical entities in a network element shall be provisionable via configuration management.

NOTE: The value of $n_{max}$ is protection application dependent.

The extra traffic signal (supported in 1:n architectures only) shall be referred to as number $n_{max}+1$. The extra traffic signal shall be transported over the protection trail/connection when this one is not transporting a normal signal and the protection trail/connection is not "locked out".

The null signal, present in 1:n architectures only, shall be referred to as number "0". When none of the normal signals nor an extra traffic signal is transported over the protection trail/connection, the null signal shall be transported. This can be any signal (e.g. one of the normal signals, a test signal, an all-ONEs signal).

### A.6 Priority of request types (conditions, external commands, states)

A request can be a local or remote:

1) condition (SF and SD) associated with a working or protection trail/connection. A condition has high or low priority.
2) state (wait-to-restore, do not revert, no request, reverse request) of the protection process.
3) external request (lockout of protection trail/connection, forced or manual switch of normal/extra traffic signal, exercise).

The basic priorities of the requests shall be as specified by table A.4. In addition, a SF-H or SF-L condition of the protection trail/connection has priority over FSw when an APS signal is supported.

NOTE: Requests are selected from the table, depending on the protection switching arrangements; i.e. in any particular case, only a subset of the requests may be required.

**Table A.4: Request Type (RT) priority**

| Request Type with APS | Request Type without APS | Priority |
|---|---|---|
| LO | LO | highest |
| *SF-H, SF-L on protection trail/connection* | - | \| |
| FSw | FSw | \| |
| SF-H | SF-H | \| |
| SF-L | SF-L | \| |
| SD-H | SD-H | \| |
| SD-L | SD-L | \| |
| MSw | MSw | \| |
| WTR | WTR | \| |
| EXER | EXER | \| |
| RR | RR | \| |
| DNR | DNR | \| |
| NR | NR | \| |
| INV | - | lowest |

### A.7 APS signal definition

#### A.7.1 APS signal fields

An automatic protection switch (APS) signal performs the communication function between the protection processes at the two ends of the protection span. For a linear protection application the following information will be passed:

- request type (RT);
- request signal number (RSN);
- local bridged signal number (LBSN);
- local architecture type (ARCH) (application dependent).

**RT:** 4 bits indicate the type of request, as listed in table A.5.

**Table A.5: Request Type mapping into APS signal**

| RT | code in RT field [MSB-LSB] |
|---|---|
| NR | 0000 |
| DNR | 0001 |
| RR | 0010 |
| EXER | 0100 |
| WTR | 0110 |
| MSw | 1000 |
| SD-L | 1010 |
| SD-H | 1011 |
| SF-L | 1100 |
| SF-H | 1101 |
| FSw | 1110 |
| LO | 1111 |

**RSN:** M bits[1] indicate the number of the signal (normal, extra, trail, connection) for which the request is issued, as shown in table A.6. The coding in the RSN field of the APS signal is binary.

**Table A.6: Request signal number**

| Signal number | Refers to requesting switch action for |
|---|---|
| 0 | Null signal or protection trail/connection signal depending on associated Request Type (RT):<br>- Conditions (SF, SD) and associated priority apply to the protection trail/connection signal.<br>- External commands (LO-#0, FSw-#0, MSw-#0) apply to the protection trail/connection.<br>- States (NR-#0, RR-#0, WTR-#0, DNR-#0) for further study. |
| 1 to $n_{max}$ | Normal signal or working trail/connection signal depending on associated Request Type (RT):<br>- Conditions (SF, SD) and associated priority apply to the corresponding working trail signals.<br>- External commands (FSw-#i, MSw-#i, EXER-#i) apply to the corresponding normal signals.<br>- States (NR-#i, RR-#i, WTR-#i, DNR-#i) for further study.<br><br>For 1 + 1, only normal signal/working trail/connection signal 1 is applicable with fixed high priority |
| $n_{max}$+1 | Extra traffic signal:<br>- Conditions (SF, SD) are not applicable.<br>- External commands (FSw-#M, MSw-#M, EXER-#M) apply to the extra traffic signal.<br>- States (NR-#M) for further study.<br><br>Exists only when provisioned in a 1 : n architecture. |

[1] M is application dependent.

**LBSN:** M bits (M is application dependent) indicate the number of the signal (null, normal, or extra) that is bridged to the protection trail, as shown in table A.7. The coding in the LBSN field of the APS signal is binary.

**Table A.7: Local bridged signal number**

| Signal number | Indication of |
|---|---|
| 0 | Null signal. |
| 1 to $n_{max}$ | Normal signal.<br><br>NOTE - For 1 + 1, only normal signal 1 is applicable. |
| $n_{max}$+1 | Extra traffic signal.<br><br>NOTE - Exists only in a 1 : n architecture. |

**ARCH:** 1 bit indicates the type of the architecture as shown in table A.8:

**Table A.8: architecture type**

| ARCH | Architecture type |
|---|---|
| 0 | 1 + 1 |
| 1 | 1 : n |

### A.7.2 STM-N MS-APS

The APS signal for 1+1 and 1:n linear STM-N MS protection consists of 13 bits organised in 4 groups as depicted in figure A.3. Refer to prETS 300 746 [6].

| K1 | | | | | | | | K2 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 |
| request type | | | | request signal number | | | | local bridged signal number | | | | arch |

**Figure A.3: STM-N MS-APS definition**

### A.7.3 STM-N VC-APS

VC APS definition and bit allocation is for further study.

**Figure A.4: VC APS definition (to be defined)**

## A.8 Switch performance: switching and holdoff times

For automatically initiated conditions (i.e. SF and SD), the protection switch completion time shall be less than 50 ms. Protection switch completion time excludes the detection time necessary to initiate the protection switch, and hold-off time. It includes the transmission transfer delay time when bi-directional and 1:n uni-directional switching is selected.

NOTE 1: The allocation of the maximum protection switching completion time is currently under study in ITU-T.

NOTE 2: When bi-directional and 1:n uni-directional switching is required, the transfer delay time may limit the length of the protected trail/connection. This is due to the transfer delay of protection information that is to be communicated between the two ends via the APS signals. Alternatively, the protection switch time for such a case could be defined as a value with 3 components: a fixed (basic) value, the length of the protection trail/connection, and the number of network elements and their processing level (e.g. AU only, AU and TU). This is for further study.

Hold-off times are useful to stagger protection switching activation between various transport layers or within the same transport layer. It shall be possible to provision for each protection group (refer to ETS 300 417-1-1 [1], subclause 9.2.1) whether or not a holdoff timer is enabled. The objective is that the holdoff time should be selectable per protection group on an individual basis. As a minimum, a single holdoff time per layer shall be supported, applicable for all protection groups within that layer. The defect condition should be continuously monitored for the full duration of the hold-off time before switching occurs. The hold-off time should therefore be provisionable from 0 to 10 seconds in steps of 100 ms.

NOTE 3: The specification of the operation of a holdoff timer within the protection switch process is for further study.

The service interruption due to the switching on an external command (CLR, LO, FSw, MSw) shall be limited to the switch-over time.

### A.9 Subprocesses

This subclause specifies in a more or less formal manner the operation of the subprocesses within the protection process.

NOTE 1: SDL specification for the following pseudo code is for further study.

NOTE 2: The addition of a Lockout of Working #i command is for further study.

NOTE 3: The addition of a holdoff timer is for further study.

**Signal request (type & signal number) processes**

This process shall transfer the input SF and SD signals from a trail/connection (either protection (#0), or working #1, .. , or working #n) into a Signal Request Type (SRT) and Signal Request Signal Number (SRSN):

- The SRSN shall be "0" (zero) for the protection trail/connection and "i" ($1 \leq i \leq n$) for working trail/connection #i.

- The SRT shall be generated based on the inputs SF, SD, SFpriority, SDpriority, as follows:

```
if (SF==true)
then  if (SFpriority==high)
      then SRT= SF-H
      else SRT=SF-L
      fi
else  if (SD==true)
      then if (SDpriority==high)
           then SRT=SD-H
           else SRT=SD-L
           fi
      else SRT= NR
      fi
fi
```

**External request (type & signal number) process**

This process shall transfer the external commands (EXTCMD) into an External Request Type (ERT) and External Request Signal Number (ERSN):

- The ERSN shall be "0" (zero) if no normal signal is indicated, "i" ($1 \leq i \leq n_{max}$) for normal signal #i, and "$n_{max}$+1" for the extra traffic signal
- The ERT/ERN shall be generated as follows:

```
do on external command reception
    start 2.5 s Completion Timer (CTimer)
    if (EXTCMD==clear)
    then ERT=NR
        ERSN=0
    else if (EXTCMD==lockout of protection)
        then  ERT=LO
              ERSN=0
        else  if (EXTCMD==forced switch-#i)
              then  ERT=FSw
                    ERSN=#i
              else  if (EXTCMD==manual switch-#i)
                    then  ERT=MSw
                          ERSN=#i
                    else  if (EXTCMD==exercise-#i)
                          then  ERT=EXER
                                ERSN=#i
                          fi
                    fi
              fi
        fi
    fi
    wait until CTimer is expired
    then {check if FSw request is denied, then release external (FSw) request}
        if (ERT==FSw) and not [ ((GRT==FSw) or (GRT==RR)) and (GRSN==ERSN) ]
        then  ERT=NR
              ERSN=0
        fi
        {check if LO request is denied, then release external (LO) request}
        if (ERT==LO) and not [((GRT==LO) or (GRT==RR)) and (GRSN==ERSN)]
        then  ERT=NR
              ERSN=0

        {check if MSw request is denied, then release external (MSw) request}
        if (ERT==MSw) and not [ ((GRT==MSw) or (GRT==RR)) and (GRSN==ERSN) ]
        then  ERT=NR
              ERSN=0
        fi
        {check if EXEC request is denied, then release external (EXEC) request}
        if (ERT==EXEC) and not [((GRT==EXEC) or (GRT==RR)) and(GRSN==ERSN)]
        then  ERT=NR
              ERSN=0
        fi
    tiaw
od
```

NOTE 4: the above clearing of external requests is continuously active after expiry of 2.5 seconds timer. If the external command was acknowledged initially, but is overruled later on, the external command is dropped consequently.

**Local request (type & signal number) priority process**

This process shall determine the highest priority local request. It shall evaluate the status of the protection and working input signals (SRT/SRSN #0 to SRT/SRSN #n), the external command (ERT/ERSN), and protection parameters OPERtype and EXTRAtraffic by a three step priority logic:

1) The highest priority local request shall be determined over the set of SRT/0, SRT/1, .., SRT/n, ERT inputs based on the descending order of request type priorities in table 0;

2) If there is at least one SRT that is higher than the ERT, and if two or more trails/connections (working/protection) have the same highest request type (SRT), the trail/connection with the lowest number shall take priority, unless the priority of the highest SRT is identical to the current LRT.

NOTE 5: The protection trail/connection has the highest priority due to its number (#0).

NOTE 6: When normal signal number B is already transported via the protection trail/connection, it will not be replaced by normal signal number A (A < B) if both working trail/connection A and working trail/connection B have the same defect condition with the same priority (i.e. SRT/A == SRT/B is e.g. SF-H).

3) If highest priority request (SRT, ERT) detected under 1. and 2. is no-request (NR), the LRT depends on the history of the protection process, the operation type, and the presence of an extra traffic signal.

The following pseudo code describes this 3 step process:

```
if ((LRT==WTR) and (ERT==EXER))
then                    {exercise command is of lower priority then wait-to-restore state}
    LRTnew=NR
    LRSNnew=0
    LRsource=signal
else
    LRTnew = ERT            {initialise process}
    LRSNnew=ERSN
    LRsource=external
fi
for i==0 to n
do                          {find highest priority local request active}
    if (LRTnew < SRT/i)
    then LRTnew=SRT/i
        LRSNnew=i
        LRsource=signal
    fi
od
if (LRTnew==NR)
then                        {No-Request case}
    if (OPERtype==non-revertive)
    then                    {non-revertive case}
        if (LRSN==1)        {check if do not revert needs to be generated}
        then LRT= DNR
        else LRT=NR
        fi
    else                    {revertive case}
        if ( ((LRT==SF) or (LRT==SD) or (LRT==WTR)) and (WTRtimer > 0) )
        then        {previous request was a SF or SD, or a WTR running}
            LRT=WTR
        else                {previous request was no-request}
            LRT=NR
            if (EXTRAtraffic==true)
            then        {extra traffic supported}
```

```
                                LRSN=n_max+1
                        else    {extra traffic not supported}
                                LRSN=0
                        fi
                fi
        fi
else                            {Request case}
        if (LRsource==external)
        then                    {external local request has highest priority}
                LRT=LRTnew
                LRSN=LRSNnew
        else                    {a signal has highest local request priority}
                if (LRTnew≠LRT)
                then            {new request not equal to existing local request}
                        LRT=LRTnew
                        LRSN=LRSNnew
                else            {new request equal to existing local request}
                        if (SRT/LRSN≠LRTnew)
                        then    {existing local request source has changed request}
                                LRSN=LRSNnew
                        fi
                fi
        fi
fi
```

In revertive mode of operation a wait-to-restore timer (WTRtimer) shall be supported. The wait-to-restore period (WTRtime) shall be provisionable in the order of 0 - 12 minutes, in steps of Y seconds. The timer shall be set to the provisioned value when the SF or SD defect condition is active (LRT=SF or LRT=SD). The timer shall be started when the last defect condition (SF, SD) clears; i.e. when all SRTs indicate No Request (NR). The WTR timer shall count down to zero. The WTR timer shall be reset to zero (deactivates earlier) if the Global Request Type (GRT) no longer indicates wait-to-restore, i.e. when any request of higher priority pre-empts this state.

The value of Y is for further study.

**APS interpretation process**

This process shall translate the accepted APS signal into the signals Remote Request Type (RRT), Remote Request Signal Number (RRSN), Remote Bridged Signal Number (RBSN) and Remote Architecture type (RARCH), as follows:

- RRT as specified in table A.9.
- RRSN = AcRSN
- RBSN = AcLBSN
- RARCH = AcARCH.

NOTE 6: AcRSN and AcLBSN can be out of range due to a fault or bit errors. For such case an invalid defect will be detected. See below.

**Table A.9: Remote Request Type (RRT) interpretation from APS signal**

| AcRT | RRT |
|---|---|
| 0000 | NR |
| 0001 | DNR |
| 0010 | RR |
| 0011 | invalid |
| 0100 | EXER |
| 0101 | invalid |
| 0110 | WTR |
| 0111 | invalid |
| 1000 | MSw |
| 1001 | invalid |
| 1010 | SD-L |
| 1011 | SD-H |
| 1100 | SF-L |
| 1101 | SF-H |
| 1110 | FSw |
| 1111 | LO |

**Defects:**

If the received APS Architecture (RARCH) value differs from the local architecture type (ARCHtype) for a period of 50 ms, a Protection Architecture Mismatch defect (dPAM) shall be detected. The defect shall be cleared when the there is a match again.

If the request type bits (RT) in the APS signal indicate an invalid request code, or the RSN or LBSN indicate a non-existing trail/connection/normal signal number, an invalid command defect (dINV) shall be detected when the condition exist for 50 ms. The defect shall be cleared when the RT indicate a valid code and the RSN or LBSN indicate an existing signal number. Neither shall be considered remote requests for a locally locked out normal signal.

**Global request priority process**

The local request (LRT,LRSN) and the remote request (RRT,RRSN) shall be compared to decide which has priority. The priority shall be determined according to the descending order of priorities in table A.4. Note that a received reverse request shall not be considered in the comparison.

The result, Global Request Type (GRT) and Global Request Signal Number (GRSN) shall be determined as follows:

```
if ( (SWtype==bi-directional) and (SRT/0≠SF) and (RRT≠RR) and
      [ (RRT>LRT) or
       ((RRT==LRT) and (GRT==RR)) or
       ((RRT==LRT) and (GRT≠RR) and (RRSN<LRSN)) ] )
then                {bi-directional switching, no SF on protection trail/connection, no reverse
                     request, and either “remote request overrules local request” or “remote
                     request equals local request and was already accepted” or “remote
                     request equals local request and remote signal number is lower than
                     local signal number”}
      GRT=RR
      GRSN=RRSN
else                {uni-directional switching or SF on protection trail/connection or reverse
                     request received or local request overrules remote request or local and
                     remote requests are equal and local signal number is less or equal
                     remote signal number}
      GRT=LRT
      GRSN=LRSN
fi
```

NOTE 7: Refer to subclause A.4.

**Defects:**

If a head end response on a tail end request does not comply to the protocol (i.e. “not ((RRT==RR and RRSN==GRSN) or RRT≥LRT)”) within a period of 50 ms, an acknowledge timeout defect (dTMOUT) shall be detected. The defect shall be cleared when the head-end response complies again or if the protection trail/connection is in SF condition.

**Bridge control process**

This process controls which of the normal/extra traffic signals is bridged to the protection trail/connection. Its operation shall be as follows:

```
if (ARCHtype==1+1)
then                {1+1 architecture}
     LBSN=1         {normal #1 signal permanent bridged}
else                {1:n architecture}
     if [ (SRT/0≠SF) and (not (dPAM or dSCM or dTMOUT or dINV)) ]
     then           {no SF on protection and no failure of protocol}
          LBSN=RRSN
     else           {SF on protection or failure of protocol}
          if (SWtype==bi-directional)
          then LBSN=0
          fi
     fi
fi
```

NOTE 8: When the protection trail/connection is not in use, null signal is indicated on both RSN and LBSN fields in the APS signal. Any normal signal may be bridged to the protection trail/connection at the head end. The tail end must not assume or require any specific signal.

**Selector control process**

This process controls which of the normal/extra traffic signals is connected to/extracted from the protection trail/connection. Its operation shall be as follows:

```
if ( (ARCHtype==1+1) and (SWtype==uni-directional) )
then
     if [ (SRT/0≠SF) or (APSmode==false) ]
     then LSSN=LRSN
     else LSSN=0        {release the selector due to SF on protection when an APS
                         channel is in use}
     fi
else                    {1+1 bidirectional switching or 1:n uni- & bi-directional switching}
     if [(GRSN==RBSN) and (SRT/0≠SF) and (not (dPAM or dSCM or dINV or dTMOUT))]
     then LSSN=GRSN
     else LSSN=0        {release the selector due to protection SF or failure of protocol}
     fi
fi
```

NOTE 9: In 1 + 1 architecture in uni-directional switching, each end operates independently of the other end, and APS signal is not needed to co-ordinate switch action. However, for the case an APS is supported it is still used to inform the other end of the local action.

NOTE 10: Note that selectors can be temporarily released when normal signal #i gets replaced by normal signal #j, due to temporary signal number mismatch on GRSN (RSN in transmitted APS signal) and RBSN (LBSN in received APS signal).

NOTE 11: The operation of 1 + 1 bi-directional switching is optimized for a network in which 1 : n protection switching is widely used and which is therefore based on compatibility with a 1 : n arrangement. Since the bridge is permanent, i.e. normal signal number 1 is always bridged, normal signal 1 is indicated on the LBSN field in the transmitted APS signal, unless the RSN field in the received APS signal indicates null signal (0). Switching is completed when both ends select the signal, and may take less time because LBSN indication does not depend on a bridging action.

NOTE 12: When the switch is no longer required, e.g. the failed working trail/connection has recovered from the fault and Wait-to-restore has expired, the tail end indicates No Request for Null Channel on the APS fields RT and RSN. This releases the selector due to signal number mismatch. The head end then releases the bridge and replies with the same indication on its RT and RSN fields and Null signal indication on LBSN. The selector at the head end is also released due to mismatch. Receiving Null signal on RSN causes the tail end to release the bridge. Since the LBSN fields now indicate Null Channel which matches the Null Channel on the RSN bytes, the selectors remain released without any mismatch indicated, and restoration is completed.

**Defects:**

If a mismatch between RBSN and GRSN persists for 50 ms, a Selector Control Mismatch defect (dSCM) shall be detected. The dSCM shall be cleared when RBSN is identical to GRSN or if the protection trail/connection is in SF condition.

**Consequent Actions:**

The selector shall be released if one or more of the four defects dPAM, dSCM, dTMOUT, dINV is active.

**APS generation process**

This process shall translate the signals Global Request Type (GRT), Global Request Signal Number (GRSN), Local Bridged Signal Number (LBSN) and local Architecture type (ARCHtype) into a transmitted APS signal, as follows:

- TxRT as specified in table A.10
- TxRSN = GRSN
- if ( (RRSN==0)
  then TxLBSN = 0
  else TxLBSN = LBSN
  fi
- if (ARCHtype==1+1)
  then TxARCH = 0
  else TxARCH = 1
  fi

**Table A.10: Global Request Type (GRT) mapping into APS signal**

| GRT | TxRT |
|---|---|
| NR | 0000 |
| DNR | 0001 |
| RR | 0010 |
| EXER | 0100 |
| WTR | 0110 |
| MSw | 1000 |
| SD-L | 1010 |
| SD-H | 1011 |
| SF-L | 1100 |
| SF-H | 1101 |
| FSw | 1110 |
| LO | 1111 |

**Reporting**

The issue of reporting is for further study. However, initial thoughts on this topic are given below:

The function reports the active external request, active local request, active remote request (if APS supported), reason of denial of an external command, and the condition (SF,SD) of the working and protection trails/connections.

The condition of the working and protection trails/connections is reported to present a complete set of information to allow unambiguous interpretation of the status of the protection entity and reaction on external commands.

MI_SignalStatus/i ← SRT/i

**Defect Correlations:**

cFOP ← (dSCM or dPAM or dTMOUT or dINV) and (not CI_SSF)

**Performance Monitoring:**

Every second the number of Protection Switch actions within that second shall be reported as pPSC (Protection Switch Count).

For the case of revertive operation, every second that the normal signal #i is not selected from the working trail/connection #i shall be reported as a pPSD/i, i ≥ 1, (Protection Switch Duration for working trail/connection #i). Every second that a normal signal is selected from the protection trail/connection shall be reported as a pPSD/0.

# Annex B

## The PROMELA Model of the ETSI Protection Switching Process

```
/*************************************************************************

  This model implements Protection Switching specifications from:
    Final Draft ETSI ETS 300 417-3-1 annex A

 *************************************************************************/

/*************************************************************************/
/* time constants, commands and variables */

/* Global variables: */
byte time=0 ;
byte nmbActiveTimers=0 ;

/* Constants: */
#define INACTIVE 255
#define MAXTIME (INACTIVE-1)

/* Timer Macros: */
#define SETTIMER(X,Y) { if :: (X==INACTIVE) -> nmbActiveTimers++ \
                  :: else -> skip fi ; X=time+(Y) }
#define CLEARTIMER(X) { if :: (X!=INACTIVE) -> nmbActiveTimers-- \
                   :: else -> skip fi ; X=INACTIVE }
#define TIMEREXPIRY(X)(time==X)
#define TIMERACTIVE(X)(X!=INACTIVE)

#if 0
/* Time Macros: */
#define WAITFORTIME(X){nmbActiveTimers++;(time==(X));nmbActiveTimers--}
             /* DONOT use as a guard!! x++ is always executable.. */

active proctype GlobalTime()
{    do
     :: timeout ->
          if
          :: (nmbActiveTimers > 0) && (time < MAXTIME) ->
               atomic
               {    time++ ;
                    printf("# time=%d\n", time)
               }
          :: (time==MAXTIME) ->
               printf("# maxtime (%d) reached -> stop\n", MAXTIME) ;
               break
          :: else ->
               printf("# done.\n");
               time=MAXTIME ; /* allow other to see time is over */
               break
          fi
     od
}
#endif
/*************************************************************************/

/* 0=disable, 1=include statements for printing */
#define PRINTAPS 1
#define PRINTIS 0

/* The missing logical operator (X->Y) */
#define imply(X,Y) ((!(X)) || (Y))

/* define soft_assert: (0) PROMELA assert or (1) only print 'ugh!' */
#if 0
#define soft_assert(X) {if :: !(X) -> printf("# -ugh!-\n") :: else -> skip fi}
#else
#define soft_assert(X) assert(X)
#endif

/* Globals */
#define true  1
#define false 0
#define NONE 254
#define error assert(0)

/* Parameters */

 byte Nmax ;  /* prot=0, work=1,..,Nmax, extra=Nmax+1 */
 byte ARCHtype[2] ;
#define _1_1   0
#define _1_N   1
```

```
 byte SWtype[2] ;
#define uni_directional  0
#define bi_directional   1
 byte OPERtype[2] ;
#define revertive      0
#define non_revertive  1
 bool APSmode[2] ;
 int  WTRtime ;
 int  SwitchingTime ;
 byte PROTtype ;
#define SNC   0
#define TRAIL 1
 bool EXTRAtraffic[2] ;

#define N1 0
#define N2 1

         /* channel priorities (used in external request) */
#define LOW  31
#define HIGH 30

         /* channel status */
#define SF   22  /* link (or upstream link) has signal fail */
#define SD   21  /* link (or upstream link) has signal degrade */


#define CLEAR 14  /* External command */
#define TO    13  /* signal from timer-expiry */

/* Request Types, with priorities */
#define LO    12
         /* both SFL and SFH on Prot is higher than SF on Working and FSw */
#define FSw   11
#define SFH   10
#define SFL   9
#define SDH   8
#define SDL   7
#define MSw   6
#define WTR   5
#define EXER  4
#define RR    3
#define DNR   2
#define NR    1
#define INV   0       /* invalid */

#define SOURCE_SIGNAL   0
#define SOURCE_EXTERNAL 1

mtype = {
         /* protection channel msg-type */
STATUS,  /* msg-type is channel status (NR/SF/SD)  */
APS      /* msg-type is APS */
}

byte etsi_lssn[2];
byte etsi_lbsn[2];
byte etsi_grt[2] ;

byte soon=0 ;

/*************************************************************************/

proctype  ETSInode(byte n;
chan extcmd, in1, out1, in2, out2, win1, wout1, win2, wout2,protin, protout;
byte HOtime ;
byte     SDpriority0, SDpriority1, SDpriority2,
     SFpriority0, SFpriority1, SFpriority2 )

{    byte HOtimer=INACTIVE ;

     byte cmd, extsig ;

     byte lrtnew, lrsnnew ;
     byte i;
     byte lrsource ;

     byte tx_lbsn ;

     byte prev_grt=NR ;
```

```
    bool stimul_is_aps ;
    byte msg ;
    byte m1, m2, m3 ;
    byte prev_m1=NR, prev_m2=0, prev_m3=0  ;
    bool newAPS ;
    byte sr ; /* signal request number */

byte lrt=NR ;
byte lrsn=0 ;
byte lbsn=0 ;
byte lssn=0 ;

byte grt=NR ;
byte grsn=0 ;

byte rrt=NR ;
byte rrsn=0 ;
byte rbsn=0 ;

byte ert=NR ;
byte ersn=0 ;

byte WTRtimer=INACTIVE ;

byte srt[3] ;
byte SDpriority[3] ;
byte SFpriority[3] ;

/* defect = (dPAM or dSCM or dINV or dTMOUT) */
bool defect=false ;

INITIALIZE:
atomic { /* ------------------------------------------------------------- */
    srt[0]=NR; srt[1]=NR; srt[2]=NR;
    SDpriority[0]=SDpriority0 ; SFpriority[0]=SFpriority0 ;
    SDpriority[1]=SDpriority1 ; SFpriority[1]=SFpriority1 ;
    SDpriority[2]=SDpriority2 ; SFpriority[2]=SFpriority2 ;
    if
    :: ARCHtype[n]==_1_1 ->
        lbsn = 1
    :: else -> skip
    fi;
    etsi_lssn[n] = lssn ; /* export local variables */
    etsi_lbsn[n] = lbsn ;

#if PRINTAPS
    printf("# Parameters ETSInode(%d): ",n+1);
    if
    :: ARCHtype[n]==_1_1      -> printf("1+1, ")
    :: ARCHtype[n]==_1_N      -> printf("1:N, ")
    fi ;
    if
    :: SWtype[n]==uni_directional  -> printf("unidir, ")
    :: SWtype[n]==bi_directional   -> printf("bidir, ")
    fi ;
    if
    :: OPERtype[n]==non_revertive  -> printf("non-rev, ")
    :: OPERtype[n]==revertive  -> printf("revertive, ")
    fi ;
    printf("APS=%d, ", APSmode[n]);
    printf("EXTRAtraffic=%d", EXTRAtraffic[n]) ;
    printf("\n") ;
#endif

AGAIN:   skip ;

WAITMSG:
    stimul_is_aps=false ;
    /* The rest state of the node: */
end: do
    :: protin?APS,m1,m2,m3 ->
         goto APSIN
    :: extcmd?cmd,extsig ->
        goto EXTERN
    :: win1?STATUS,srt[1] ->
        sr=1;
        goto SIGNALREQUEST
    :: win2?STATUS,srt[2] ->
        sr=2;
```

```
         goto SIGNALREQUEST
    :: protin?STATUS,srt[0],m2,m3  ->
         sr=0;
         goto SIGNALREQUEST
    od ;

APSIN:   newAPS = (m1!=prev_m1 || m2!=prev_m2 || m3!=prev_m3) ;
    prev_m1=m1; prev_m2=m2; prev_m3=m3 ;
    if
    :: (srt[0]!=SFL && srt[0]!=SFH) && (APSmode[n]==true) &&
       !(ARCHtype[n]==_1_1 && SWtype[n]==uni_directional) &&
       newAPS
         ->
         rrt=m1;
         rrsn=m2;
         rbsn=m3;
         stimul_is_aps=true ;
         goto LOCAL
    :: else ->
         goto AGAIN ; /* cannot receive APS msg's */
    fi ;

EXTERN: if
    :: extsig>Nmax -> error
    :: else -> skip ;
    fi ;

    if
    :: (cmd==CLEAR) ->
         ert  = NR ;
         ersn = 0 ;
    :: (cmd==LO) ->
         if
         :: (OPERtype[n]==revertive) ->
              ert  = LO ;
              ersn = 0 ;
         :: else ->
              goto AGAIN ;
         fi ;
    :: (cmd==FSw) ->
         if
         :: (OPERtype[n]==revertive) && (extsig==0) ->
              goto AGAIN ;
         :: else ->
              ert  = FSw ;
              ersn = extsig ;
         fi ;
    :: (cmd==MSw) ->
         if
         :: (OPERtype[n]==revertive) && (extsig==0) ->
              goto AGAIN ;
         :: else ->
              ert  = MSw ;
              ersn = extsig ;
         fi ;
    :: (cmd==EXER) ->
         ert  = EXER ;
         ersn = extsig ;

    :: (cmd==TO) ->
         if
         :: (OPERtype[n]==revertive) ->
              CLEARTIMER(WTRtimer) /* WTR TO modelled via ext.cmd */
         :: else ->
              goto AGAIN ;
         fi ;

    :: else -> error
    fi ;
    goto LOCAL ;

SIGNALREQUEST:
    if
    :: (srt[sr]==SF) ->
         if
         :: (SFpriority[sr]==HIGH) ->   srt[sr]=SFH
         :: else  ->           srt[sr]=SFL
         fi
    :: (srt[sr]==SD) ->
```

```
        if
        :: (SDpriority[sr]==HIGH) ->   srt[sr]=SDH
        :: else ->            srt[sr]=SDL
        fi
    :: (srt[sr]==NR) ->            srt[sr]=NR
    :: else -> error ;
    fi ;

LOCAL:   if
    :: lrt==WTR && ert==EXER ->
        lrtnew = NR ;
        lrsnnew = 0 ;
        lrsource = SOURCE_SIGNAL ;
    :: else /* normal case: initiate new with external */
        lrtnew = ert ;
        lrsnnew = ersn ;
        lrsource = SOURCE_EXTERNAL
    fi ;

    i = 0 ;
    do
    :: i>Nmax -> break
    :: else ->
        if
        :: (srt[i] > lrtnew) ->
            lrtnew = srt[i] ;
            lrsnnew = i ;
            lrsource = SOURCE_SIGNAL
        :: else -> skip
        fi ;
        i++ ;
    od ;

    if /* SF/0 - FSw prio */
    :: (lrtnew==FSw) && ((srt[0]==SFH) || (srt[0]==SFL)) && APSmode[n] ->
        lrtnew = srt[0] ;
        lrsnnew = 0 ;
        lrsource = SOURCE_SIGNAL
    :: else -> skip
    fi ;

    if
    :: lrtnew==NR -> /* Non-request case */
        if
        :: OPERtype[n]==non_revertive /* => ARCHtype[n]==_1_1 */ ->
            if
            :: (grsn==1) ->
                lrt = DNR
            :: else ->
                lrt = NR ;
                lrsn = 0
            fi
        :: OPERtype[n]==revertive ->
            if
            :: lrt==SFH ||
               lrt==SFL ||
               lrt==SDH ||
               lrt==SDL    ->
                lrt = WTR ;
                SETTIMER(WTRtimer, WTRtime) ;
            :: lrt==WTR && TIMERACTIVE(WTRtimer)
                    && !TIMEREXPIRY(WTRtimer) ->
                lrt = WTR ;
            :: else ->
                lrt = NR ;
                CLEARTIMER(WTRtimer) ;
                if
                :: EXTRAtraffic[n]==true ->
                    lrsn = Nmax+1
                :: else ->
                    lrsn = 0
                fi
            fi
        fi
    :: else /* lrtnew!=NR */ ->         /* Request-case */
        if
        :: lrsource==SOURCE_EXTERNAL ->
            lrt=lrtnew ;
            lrsn=lrsnnew
```

```
        :: lrsource==SOURCE_SIGNAL ->
            if
/*          :: lrtnew!=lrt || ( ((srt[0]==SFH)||(srt[0]==SFL)) && APSmode[n]) ->*/
            :: lrtnew!=lrt ->
                lrt=lrtnew ;
                lrsn=lrsnnew
            :: else -> /* same request type ... */
                if
                :: srt[lrsn] != lrtnew -> /* only if */
                    lrsn = lrsnnew
                :: else -> skip
                fi
            fi
        fi
    fi ;

GLOBAL:  skip ;

    soon++ ;
    assert(soon < 20) ;
/* within X passes through this point (including all processes) the protocol */
/* must stabilize. */

    if
    ::  SWtype[n]==bi_directional &&
        !((srt[0]==SFH) || (srt[0]==SFL)) &&
        rrt!=RR &&
        (rrt>lrt ||
         (rrt==lrt && rrt!=NR &&
          (grt==RR ||
           (grt!=RR && rrsn<lrsn)
        ))) ->
            grt = RR ;
            grsn = rrsn ;
    :: else ->
            grt = lrt ;
            grsn = lrsn ;
    fi ;
    if
    ::  SWtype[n]==bi_directional &&
        lrt==FSw &&
        ( ((rrt==SFH) || (rrt==SFL)) && (rrsn==0) ) ->
            grt = RR ;
            grsn = rrsn ;
    :: else ->   skip
    fi ;

REVIEWEXTERN:
    if
    :: (ert!=NR) &&
       !( (grt==ert || grt==RR) && grsn==ersn ) ->
        ert = NR ;
        ersn = 0 ;
        goto LOCAL ; /* do local again */
    :: else -> skip ;
    fi;

CHECKWTR:
    if
    :: lrt==WTR && grt!=WTR ->
        lrt=NR ;
        CLEARTIMER(WTRtimer) ;
/*      goto LOCAL /* do local again */
    :: else -> skip
    fi ;

BRIDGE:  if
    :: ARCHtype[n]==_1_1 ->
        lbsn = 1 ;
    :: else ->
        if
        :: !((srt[0]==SFH) || (srt[0]==SFL)) && !defect ->
            lbsn = rrsn ;
        :: else ->
            if
            :: (SWtype[n]==bi_directional) ->
                lbsn = 0
            :: else ->
                lbsn = lbsn ;
```

```
            fi
        fi
    fi ;

SELECTOR:
    if
    :: ARCHtype[n]==_1_1 && SWtype[n]==uni_directional ->
        if
        :: ((srt[0]==SFH) || (srt[0]==SFL)) && APSmode[n]==true ->
            lssn = 0
        :: else ->
            lssn = grsn
        fi
    :: else ->
        if
        :: (grsn==rbsn) && !((srt[0]==SFH) || (srt[0]==SFL)) && (!defect) ->
            lssn = grsn
        :: else ->
            lssn = 0
        fi
    fi ;

TRANSMIT:
    if
    :: rrsn==0 ->
        tx_lbsn = 0
    :: else ->
        tx_lbsn = lbsn
    fi ;

    etsi_lssn[n] = lssn ; /* export local variables */
    etsi_lbsn[n] = lbsn ;
    etsi_grt[n] = grt ;

PRINTSTATUS: skip ;

#if PRINTAPS
    byte rt;
    if
    :: (!newAPS) -> skip/*m1 = NONE*/
    :: else -> skip
    fi ;
    printf("# |  %d |  %d | ", etsi_lssn[N1], etsi_lbsn[N1]) ;
    rt = ((n==N1)->grt:NONE) ;
    if
    :: rt==LO   -> printf("LO  ")
    :: rt==FSw  -> printf("FSw ")
    :: rt==SFH  -> printf("SFH ")
    :: rt==SFL  -> printf("SFL ")
    :: rt==SDH  -> printf("SDH ")
    :: rt==SDL  -> printf("SDL ")
    :: rt==MSw  -> printf("MSw ")
    :: rt==WTR  -> printf("WTR ")
    :: rt==EXER -> printf("EXER")
    :: rt==RR   -> printf("RR  ")
    :: rt==DNR  -> printf("DNR ")
    :: rt==NR   -> printf("NR  ")
    :: rt==INV  -> printf("INV ")
    :: rt==NONE -> printf("    ")
    :: else     -> printf("huh?")
    fi ;
    if
    :: (rt==NONE) ->  printf("    ") ;
    :: else ->        printf(",%d,%d", grsn,tx_lbsn);
    fi ;
    if
    :: (n==N1) ->printf(" > ")
    :: (n==N2) ->printf(" < ")
    :: else ->   printf(" ? ")
    fi ;
    rt = ((n==N2)->grt:NONE) ;
    if
    :: rt==LO   -> printf("LO  ")
    :: rt==FSw  -> printf("FSw ")
    :: rt==SFH  -> printf("SFH ")
    :: rt==SFL  -> printf("SFL ")
    :: rt==SDH  -> printf("SDH ")
    :: rt==SDL  -> printf("SDL ")
    :: rt==MSw  -> printf("MSw ")
```

```
    :: rt==WTR  -> printf("WTR ")
    :: rt==EXER -> printf("EXER")
    :: rt==RR   -> printf("RR  ")
    :: rt==DNR  -> printf("DNR ")
    :: rt==NR   -> printf("NR  ")
    :: rt==INV  -> printf("INV ")
    :: rt==NONE -> printf("    ")
    :: else     -> printf("huh?")
    fi ;
    if
    :: (rt==NONE) ->  printf("    ") ;
    :: else ->        printf(",%d,%d", grsn,tx_lbsn);
    fi ;
    printf(" |  %d |  %d | ", etsi_lssn[N2], etsi_lbsn[N2]) ;
    printf("\n") ;
#endif

#if PRINTIS
    printf("##IS: NODE=%d, LSSN=%d, LBSN=%d\n", n, etsi_lssn[n], etsi_lbsn[n]) ;
#endif

    protout!APS,grt,grsn,tx_lbsn ;

    goto AGAIN ;
} /* end atomic ------------------------------------------------------------ */

} /* end proctype ETSInode */

chan eq = [20] of {byte, byte, byte} ;

chan extcmd_[2] = [0] of { byte, byte } ;

/*  chan in1_[2] = [0] of { mtype, byte } ;
    chan out1_[2]= [0] of { mtype, byte } ;
    chan in2_[2] = [0] of { mtype, byte } ;
    chan out2_[2]= [0] of { mtype, byte } ;
*/
chan dummy = [0] of {byte} ;

chan win1_[2]= [0] of { mtype, byte } ;
chan win2_[2]= [0] of { mtype, byte } ;
chan protin_[2]   = [0] of { mtype, byte, byte, byte } ;

proctype Random()
{   byte a,b;
    byte x=1;

AGAIN:  do
    ::  if
        :: x>3 -> goto AFTER
        :: else -> skip
        fi ;
        if
        :: a=NR;
        :: a=SD;
        :: a=SF;

        :: a=LO ;
        :: a=FSw ;
        :: a=MSw ;
        :: a=EXER ;
        fi ;
        if
        :: b=0 ;
        :: b=1 ;
        :: b=2 ;
        fi ;
        if
        :: ((x==1) || (x==3)) ->
            eq!N1,a,b ;
        :: (x==2) ->
            eq!N2,a,b ;
        fi ;
        x++ ;
    od ;
AFTER:  skip ;
}
```

```
init
{
atomic { /* ------------------------------------------------------------- */
#if PRINTAPS
     printf("# Remark: \n") ;
#endif

/*IS: stimuli */

     eq!N1,SF,1 ;
     eq!N1,SF,0 ;

/*   run Random(); */

     Nmax         = 2 ;
/* model parameters per node */
              ARCHtype[N1]            = _1_N ;
              SWtype[N1]              = bi_directional ;
              OPERtype[N1]      = revertive ;
              APSmode[N1]             = true ;
     EXTRAtraffic[N1]  = false ;
/* copy */
     ARCHtype[N2]      = ARCHtype[N1] ;
     SWtype[N2]        = SWtype[N1] ;
     OPERtype[N2]      = OPERtype[N1] ;
     APSmode[N2]       = APSmode[N1] ;
     EXTRAtraffic[N2]  = EXTRAtraffic[N1] ;

/* The differnce between N1 and N2: */
/*   SWtype[N2]        = bi_directional ; */

/* common parameters */
     WTRtime      = 1 ;
     SwitchingTime= 1 ;
     PROTtype = TRAIL ;

/* model parameter dependecies */
     byte n=N1, other;
     do
     :: n>N2 -> break ;
     :: else ->
          assert(Nmax>=1 && Nmax<=2) ;
          assert(imply(OPERtype[n]==non_revertive, ARCHtype[n]==_1_1));
          assert(imply(ARCHtype[n]==_1_N, APSmode[n])) ;
          assert(imply(ARCHtype[n]==_1_1, !EXTRAtraffic[n] && Nmax==1));
          assert(imply(SWtype[n]==bi_directional, APSmode[n])) ;

          other=((n==N1)->N2:N1) ;

run ETSInode(n,
extcmd_[n], dummy, dummy, dummy, dummy, win1_[n], win1_[other],
win2_[n], win2_[other], protin_[n], protin_[other],
1,
LOW,LOW,LOW,LOW,LOW,LOW) ;

     timeout ; /* wait */

          n++ ;
     od;

#if PRINTAPS
     printf("# |----|----|----------|----------|----|----|\n") ;
     printf("# |LSSN|LBSN| APS 1->2 | APS 1<-2 |LSSN|LBSN|\n") ;
     printf("# |----|----|----------|----------|----|----|\n") ;
#endif
     win1_[N1]!STATUS,NR ;
     timeout; soon=0 ;
     win1_[N2]!STATUS,NR ;
     timeout; soon=0 ;
     printf("# |----|----|----[initialized]----|----|----|\n") ;


     run sequential_simulator() ;

} /* end atomic ------------------------------------------------------------ */

}
```

```
/*************************************/

proctype sequential_simulator()
{
    byte node, request, signal, rt, cm ;

atomic { /* ------------------------------------------------------------- */

end: do
    :: eq?node,request,signal ->

#if PRINTAPS
         printf("### Node %d, command ", node+1) ;
         rt=request ;
         if
         :: rt==CLEAR ->       printf("CLR")
         :: rt==TO ->      printf("TO")

         :: rt==LO   ->        printf("LO")
         :: rt==FSw  ->        printf("FSw")

         :: rt==SD   ->        printf("SD")
         :: rt==SF   ->        printf("SF")
         :: rt==NR   ->        printf("NR")

         :: rt==MSw  ->        printf("MSw")
         :: rt==EXER ->        printf("EXER")
         :: else     ->        printf("huh?")
         fi ;
         if
         :: (rt!=CLEAR) && (rt!=TO) ->
             printf(" for signal %d\n", signal)
         :: else ->
             printf("\n")
         fi ;
#endif

         if
         :: (request==CLEAR) || (request==LO) || (request==FSw) ||
             (request==MSw)   || (request==EXER) ||
             (request==TO) ->
             extcmd_[node]!request,signal ;
         :: else ->
             if
             :: (signal==0) -> protin_[node]!STATUS,request,0,0
             :: (signal==1) -> win1_[node]!STATUS,request
             :: (signal==2) -> win2_[node]!STATUS,request
             :: else -> error
             fi ;
         fi ;

         timeout; soon=0 ;

soft_assert(
imply(ARCHtype[N1]==_1_N, /*->*/
                 etsi_lssn[N1]==etsi_lbsn[N2] &&
                 etsi_lssn[N2]==etsi_lbsn[N1] )
) ; /* connection */

soft_assert(
imply(SWtype[N1]==bi_directional, /*->*/
                 etsi_lssn[N1]==etsi_lssn[N2] )
); /* uniform */


#if 0
soft_assert(
imply(SWtype[N1]==bi_directional, /*->*/
         /* both grt same and not equal to RR */
    (etsi_grt[N1]!=RR && etsi_grt[N1]==etsi_grt[N2])     ||
         /* one grt is RR other is not RR and not NR */
    (etsi_grt[N1]==RR && etsi_grt[N2]!=RR && etsi_grt[N2]!=NR)    ||
    (etsi_grt[N2]==RR && etsi_grt[N1]!=RR && etsi_grt[N1]!=NR) )
); /* correct RR state */
#endif

#if 0
         /* one grt is SF on prot, same grsn is prot, lssn is prot */
    ((grt[N1]==SFL || grt[N1]==SFH) && grsn[N1]==0 && lssn[N1]==0)    ||
```

```
    ((grt[N2]==SFL || grt[N2]==SFH) && grsn[N2]==0 && lssn[N1]==0)
#endif

    od ;
} /* end atomic ------------------------------------------------------------ */
}

/*********************************/
```

# Annex C

# The SPIN Manual

# Annex D

# The modelling of time in PROMELA

**Time abstraction**

"Within ONE time unit all processes are done executing, the system is stable, and some timers or time conditions are waiting."

**Consequences**

- After a time unit all processes are blocked, either they are waiting for new input or they are waiting for a timer to expire or untill time reaches a certain value. Then a system-wide (PROMELA-) timeout occures. The time is incremented by one by the time process. If some timer/time expression becomes true the associated process can execute.

- It will not be possible to 'schedule' an event at a moment the system is still running, i.e. is not stable.

- The time process is the only process that uses timeout. Special care must be taken if timeout is used within the model also.
  Some cases:
  - It doesn't matter wether the model-timeout or a time-timeout occures; let the simulator choose (the model-timeout is either smaller or longer then some units of time).
  - If model-timeouts must occure before time progresses: use a mechanism to block the time process , e.g. proctype GlobalTime :: (!TimeIsBlocked) && timeout -> ...
  - Implement model-timeouts as timers e.g. SETIMER(to,1) ; do :: ... :: TIMEREXPIRY(to) -> CLEARTIMER(to)

**Implementation**

Note:
To force program termination when all timers are inactive, track must kept of the number of active timers (nmbActiveTimers).

```
1:   /* Global variables: */
2:   byte time=0 ;
3:   byte nmbActiveTimers=0 ;
4:
5:   /* Constants: */
6:   #define INACTIVE 255
7:   #define MAXTIME (INACTIVE-1)
8:
9:   /* Timer Macros: */
10:  #define SETTIMER(X,Y) { if    :: (X==INACTIVE) -> nmbActiveTimers++ \
11:                                :: else -> skip fi ; X=time+(Y) }
12:  #define CLEARTIMER(X) { if    :: (X!=INACTIVE) -> nmbActiveTimers-- \
13:                                :: else -> skip fi ; X=INACTIVE }
14:  #define TIMEREXPIRY(X)        (time==X)
15:  #define TIMERACTIVE(X)        (X!=INACTIVE)
16:  /* Time Macros: */
17:  #define WAITFORTIME(X)        {nmbActiveTimers++;(time==(X));nmbActiveTimers--}
18:                        /* DONOT use as a guard!! x++ is always executable.. */
19:
20:  (basic) Time process: do
21:                        :: timeout ->
22:                                if
23:                                :: (nmbActiveTimers>0) && (time<MAXTIME) ->
24:                                        time++
25:                                :: else -> break
26:                                fi
27:                        od
28:
29:  Timer commands in a example: byte aTimer ;
30:                               #define HOtime 10 ;
31:                               do
32:  - wait for expiry:           :: TIMEREXPIRY(aTimer) ->
33:  - clear a timer:                    CLEARTIMER(aTimer) ;
34:                                      ...
35:  - test a timer:              :: (expression) && !TIMERACTIVE(aTimer) ->
36:  - set a timer:                      SETTIMER(aTimer,HOtime) ;
37:                               :: (other expression) && TIMERACTIVE(aTimer) ->
38:                                      CLEARTIMER(aTIMER) ;
39:                                      break
40:                               od ;
41:  Time commands in a example:
42:  - wait until a certain time: channel!NORMAL ;
43:                               WAITFORTIME(5) ;
44:                               channel!FAIL ;
```

# Annex E

# Graduation research paper

The faculty of Electrical Engineering demands that besides the graduation report, a paper is produced that reflects the graduation work in a standard publication style.

# Linear protection switching requirements simulation & verification

J.L.R. de Graaff
Delft University of Technology
Faculty of Electrical Engineering
Telecommunications and Traffic-Control Systems Group

**Abstract—This paper presents a research in system specification and verification. The characteristics of specifications are discussed and verification languages and tools are investigated. A protocol that coordinates linear protection switching, a network availability enhancement technique, is verified and the findings are discussed.**

**Index Terms—System specification, simulation, verification, linear protection switching, APS.**

## I. Introduction

The electronic systems we design today, are becoming more and more complex. This is especially true for telecommunication systems. Generally, the behaviour and functionality in the first stages of specification, will be described in the plain English language. In a later stage, when more detailed information is necessary, a more formal way of specification semantics will be required.

A first and most important issue in specification is the specification itself. How can system behaviour be modelled? What are the characteristics of the aspects of system behaviour? What languages are needed? Can we model a system in one language or do we need different languages for different aspects of the system? And how do these languages interact?

If a system cannot be modelled adequately, problems are not far away.

The second issue is: How do we know if these specifications are correct? Do these specifications actually describe the intended behaviour? Errors in these specifications can be disastrous for the functioning of the system, either on a equipment level or on the network level. When for example an error in a protocol, on which a large public telephone network is relying, results in breaking down of all communications, thousands of users are cut off. Such catastrophes results in significant loss of revenues or even human lives.

Verification of system specifications is an absolute must. Besides the determination of correctness of specifications, the verification process also adds insight and confidence in the system.

## II. System Specifications

Describing a system is a complex task and has many pitfalls. Telecommunication systems are first designed at a functional and behavioral level. This description asks for a language that can adequately model the aspects of the system, such as the functional behavior, the behavior in time, the architectural structure. These specifications are used as a basis for further development and implementation.

To indicate the many requirements for specifications, the list below is presented.

Afstudeerdocent: ir. J.A.M. Nijhof
Begeleider: ir. F. Speelman
Technisch begeleider: ir. M.P.J. Vissers
Periode: januari 1997 - augustus 1997
Opdrachtnummer: A-786
Datum: 26 February, 2001

Successful specifications are:
A) Complete, correct, unambiguous and consistent
B) Minimal
C) Usable
D) Identifiable, traceable and surveyable
E) Verifiable and testable for properties
F) Executable
G) Easily modified
H) Linkable to other requirements

Requirement A is the fundamental purpose of specifications.
Requirement B addresses over-specification.
Requirement C addresses implementability of the specifications.
Requirement D demands that specifications can be understood and related to common terms and concepts.
Requirement E demands the specification to be testable; this requires that the specification also formulates *what exactly* is correct.
Requirement F demands the specification to be understandable by the computer.
Requirement G and H address the maintainability and transportability of specifications.

# III. System Verification

## *A. Verification methods*

To test a system for correct behaviour, its specification is verified. In principal, systems can be verified manually, with use of mathematics or some systematic method. But for larger and more complex systems this is becoming impossible. This is especially true for concurrent systems, like computer and telecommunication systems and protocols. An automated verification method is a must.

A well-known type of systems, that are often subject in verification, is the protocol. Correctness of protocols is discussed [1] and more extensively in [2].

In order to verify automatically, the use of the computer is needed and for that the system has to be described formally. The formal description of the system requires a formal language, with both a formal syntax and a formal semantics. A formal syntaxis describes the model in a precise and unambiguous way. A formal semantics assigns a precise mathematical meaning to each of the modelling primitives. See [3] for more information on this subject.

In addition to the formal modelling, a tool is needed that can execute this model and determine its correctness. Generally, verification tools convert system models into finite state machines and analyze their states.

In order to determine the correctness of system behaviour, a description of correct behaviour is also needed. These are also called correctness requirements. If *all possible* behaviour of the system model satisfies these correctness requirements, the model is said to be *validated*.

Note that system simulation does not validate its behaviour, it only shows the behaviour after a certain stimulus of the system.

A major problem in system validation is the *state explosion problem*. This denotes the fact that the total number of system states can be that large, that a complete analysis may not be possible in a certain environment. Common limitations are the available memory to store the analyzed states and the maximum time a validation may take.

A system is validated by performing a full state search analysis. The usual strategy is the depth-first search, see [2]. When such a validation stops, because a limitation is reached, the analyzed part is a ‘solid cut’ of the total state structure, as depicted in Fig. 1.

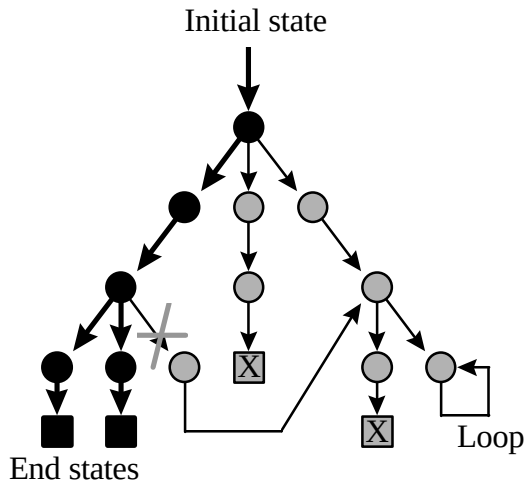



Fig. 1 State analysis

As incorrect system states tends to be equally distributed in both the depth as the breadth, a limited full state search probably only detects a small number of errors.

In such a limited environment, better search strategies can be used, that aim to analyze as much of the system as possible within the imposed limitations. This requires a *state reduction technique* or *proof approximation technique*. See Fig. 2

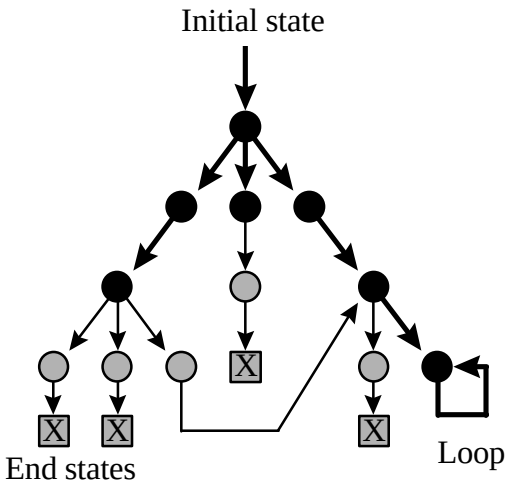



Fig. 2 Partial state analysis

*B. Verification languages and tools*

In addition to formal syntaxis and semantics, a language must possess several other properties.

In order to adequately model the behaviour of distributed systems, as is encountered much in the field of telecommunication and information technology, some required properties are:

- *Concurrency*; the independent execution of processes in an environment.
- *Synchronous/asynchronous communication*; processes communicate synchronously when the sending and receiving of a message occurs at the same time and asynchronously when the reception of a message may occur at any time. In this case, the channel has memory.
- *Non-determinism*; the possibility to specify a set of different courses of action, Leaving it open which of them is chosen.
- *Time*; time can be modelled as a physical unit (seconds, milliseconds), as logical unit (clock ticks) or as temporal properties (A occurs ‘after’ B).

*C. The tool SPIN*

SPIN is a software package that supports the formal verification of concurrent systems. The software was developed at Bell Labs in the formal methods and verification group. It is extensively discussed in [2].

SPIN has been used to trace logical design errors in distributed systems design, such as operating systems, data communications protocols, switching systems, concurrent algorithms, railway signaling protocols, etc. The tool checks the logical consistency of a specification. It reports on deadlocks, unspecified receptions, flags incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes.

To verify a design, a formal model is built using PROMELA[1], SPIN's input language. SPIN is an abbreviation of Simple PROMELA Interpreter. PROMELA is a non-deterministic

[1] Process Meta Language

language. It contains the primitives for specifying asynchronous (buffered) message passing via channels, with arbitrary numbers of message parameters. It also allows for the specification of synchronous message passing systems (rendezvous). Mixed systems, using both synchronous and asynchronous communications, are also supported.

SPIN can be used in three basic modes:

- as a **protocol simulator**, allowing for rapid prototyping with a random, guided, or interactive simulations
- as an **exhaustive state space analyzer**, capable of rigorously proving the validity of user specified correctness requirements
- as a **bit-state space analyzer** that can validate even very large protocol systems with maximal coverage of the state space (a proof approximation technique).

SPIN is very useful in verifying distributed systems and has all required properties to handle complex and large systems.

# IV. Protocol Verification

## *A. Linear Protection Switching*

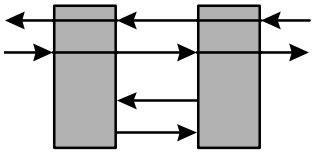
Protection switching is a method to provide a certain level of survivability in a transport network by means of diversity. Network survivability is defined as the ability to provide continuity of transport services in the presence of failures.

When transmission lines fail, their service is automatically taken over by pre-assigned redundant transmission lines. This is called *protection*.

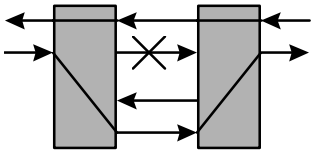
Another method to protect transmission lines is *restoration*. In this case, the redundant capacity is not pre-assigned and must be discovered by some network intelligence. Obviously this takes more time. A common requirement for completion of protection switching is 50 ms.

Today only the *linear* and the *ring* topologies are used in protection switching schemes. The ring protection topology is not discussed here.

In linear protection switching, one or more transmission lines between two certain nodes in a network are protected with a redundant transmission line, called protection line. See Fig. 3.



Normal conditions



Protection condition

Fig. 3 Linear protection switching

Linear protection switching comes in different forms. Either one or more lines can be protected with one protection line and protection can involve both transmission directions together or each direction separately. The latter is shown in Fig. 3.

The switching requires a communication (telemetry) channel to coordinate the protection switching operations. Such a channel is called *APS*[1] *channel*. The coordination is handled by the *APS protocol*.

This protocol must resolve which protection node has the highest priority request for switching to protection and then take care that both nodes switch the correct line over protection.

The APS protocol is depicted in Fig. 4. When node A detects a line failure, it sends an APS message to node B, requesting for a protection switching action. Node B responds with a message, indicating that it will follow this

[1] Automatic Protection Switching

request. Then node A sends a message that completes the switching action.

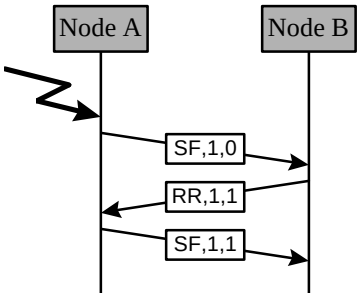



Fig. 4 Example of APS protocol behaviour

*B. The ETSI Specifications*

The linear protection switching specification in the ETSI[1] ETS300-417 standard attempts to capture the behaviour of linear protection switching in SDH[2] networks. It is based in the protection switching specification from the G.783 standard, a ITU[3] standard..

The behavioral description is specified in pseudocode, a general imperative computer language, that resembles the C or PASCAL languages.

*C. Verification results*

The verification of the APS protocol have resulted in several observations. Firstly, simulations have shown that some intended behaviour was not specified. Secondly, simulations and validations have shown that the specification was not correctly implementing the intended behaviour. Finally, some questions have arisen, concerning design choices.

All together, six design flaws were found and adjustments or corrections are proposed and verified. Also some open issues are left, that need future attention.

[1] European Telecommunications Standards Institute

[2] Synchronous Digital Hierarchy, a standard for transport networks

[3] International Telecommunication Union

The verification has added more insight in the behaviour and the specification and confidence in the specifications has grown.

# V. Conclusions

System specifications should be written as formal and unambiguous as possible to enable verification. Performing the verification on a computer is absolutely necessary, because even simple communication systems or protocols can be to large and complicated to verify manually or mathematically.

The specification and verification of systems demand several properties of the specification language and the verification tool. Note that not all languages are suitable to model all system aspects and functions adequately.

From this investigation, a tool and its related language is found to be suitable for the verification of the linear protection switching specification. This tool is called SPIN and its modelling language is called PROMELA.

This tool is easy to use and has a large validation power, which is required for large and complex systems.

The SPIN tool is used to verify the APS protocol and several flaws were found, some characteristics have come to the surface and the confidence in the specification has grown.

The reader is referred to [4] for more information on this research.

# References

[1] J. Walrand, *Communication Networks: A First Course*, Homewood, IL: IRWIN, ISBN 0-256-08864-0, 1991.

[2] G.J. Holzmann, *Design and Validation of Computer Protocols*, Englewood Cliffs, N.J.: Prentice Hall, ISBN 0-13-539925-4, 1991.

[3] P.H.A. van der Putten and J.P.M. Voeten, *Specification of Reactive Hardware/Software Systems*, Ph.D. thesis, Technical University Eindhoven, Eindhoven, The Netherlands, 1997.

[4] J.L.R. de Graaff, *Linear Protection Switching Requirements Simulation & Validation*, Graduation thesis, Document reference: A-786, Telecommunications and Traffic-Control Systems Group, Faculty of Electrical Engineering, Technical University of Technology, The Netherlands, 1997.