JAVA IMPLEMENTATION OF AX.25 LINK-LAYER PROTOCOL FOR FUTURE MICRO-SATELLITES

THETHE TSHEPO RAMONYALIOA

THESIS PRESENTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING OF THE UNIVERSITY OF STELLENBOSCH IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE MASTERS OF SCIENCE IN ENGINEERING.

SUPERVISOR: PROF. S. MOSTERT

APRIL 2003

Stellenbosch University http://scholar.sun.ac.za

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis is, unless otherwise stated, my own original work and has never been submitted in part or in its entirety at any University for any requirements towards the achievement of any degree.

T.T. Ramonyaĭioa

Stellenbosch University http://scholar.sun.ac.za

Abstract

This thesis investigates the viability of implementing the AX.25 protocol in Java, for satellite applications. The AX.25 protocol forms part of the *Communications* subsystem of a micro-satellite. It describes the implementation of a standard packet-radio link-layer communication protocol in Java, for future use on satellite on-board computers, allowing amongst other things, a reliable communications platform.

An investigation into factors that make AX.25 preferable over other communication protocols, as well as advantages of Java as a language used in the implementation, is made. The design of the implementation is described. Finally, efficiency of the Java implementation is evaluated and optimizations identified and recommended.

Stellenbosch University http://scholar.sun.ac.za

Opsomming

Hierdie tesis bespreek die lewensvatbaarheid van 'n Java implementering van die AX.25 protokol vir satelliet toepassings. AX.25 is deel van die kommunikasie stelsel van 'n mikro-satelliet. Dit beskryf die implementering van 'n standaard radio data kommunikasie intervlak in Java vir die toekomstige gebruik op die aanboord-rekenaarstelsels van satelliete. Hierdie intervlak bied, insluitende ander eienskappe, 'n betroubare kommunikasie platform.

'n Deeglike ondersoek na die faktore wat AX.25 meer aanloklik maak vir satellite toepassings is gemaak, asook hoe 'n Java implementering vergelyk met ander beskikbare tegnologie. Die geskiktheid van Java vir die implementering word ook bespeek in die lig van evaluerings wat gedoen is op die finale protokol.

Stellenbosch University http://scholar.sun.ac.za

To my Mother and in Memory of My Father

Stellenbosch University http://scholar.sun.ac.za

Acknowledgements

I would first like to thank the Lord for providing me with this opportunity to express my abilities, and for giving me the strength & courage to complete this work. I would also like to thank the following people for their varied contributions:

- My family for their continued support, patience and understanding.
- My supervisor, Prof. Sias Mostert, for believing in me and providing me with enormous academic support & for helping me immensely in my non-academic affairs; I can never thank him enough.
- Mr. Hans van der Merwe for his mentorship & guidance. I am sincerely grateful for having had the opportunity to learn from him.
- The latest inspiration in my life, my son Tsebo.

Stellenbosch University http://scholar.sun.ac.za

Contents

Stellenbosch University http://scholar.sun.ac.za

Stellenbosch University http://scholar.sun.ac.za

Stellenbosch University http://scholar.sun.ac.za

List of Figures

Stellenbosch University http://scholar.sun.ac.za

List of Tables

Stellenbosch University http://scholar.sun.ac.za

List of Abbreviations and Acronyms

ADCCP	Advanced Data Communication Control Procedure
ARPANET	Defense Advanced Research Projects Agency Network
ARRL	The American Radio Relay League
ATM	Asynchronous Transfer Mode
AX.25	Amateur X.25. A Packet Radio Protocol (Link Access)
ASCII	American Standard Code for Information Interchange
CCITT	Comite' Consultatif Internationale de Te'le'graphique et Te'le'phonique
CPU	Central Processing Unit
DCE	Data Circuit-terminating Equipment
DL	Data Link
DTE	Data Terminating Equipment
FCC	Federal Communications Commission
FCS	Frame Check Sequence
FDM	Frequency Division Multiplexing
FM	Frequency Modulation
FTP	File Transfer Protocol
FRMR	Frame Reject
HDLC	High Level Data Link Control
IF	Information Frame
IMP	Intermediate Message Processor
IPv6	Internet Protocol (version6)
ISO	International Organization for Standardization
LM	Link Multiplexer
MDL	Management Data Link
MIME	Multi-Purpose Internet Mail Extension
MNC	Metropolitan Network Controller
OBC	On Board Computer

Stellenbosch University http://scholar.sun.ac.za

OSI	Open System Interconnection
PTT	Push-to-talk
RR	Receive Ready Frame
UA	Un-numbered Acknowledgement Frame
UART	Universal Asynchronous Receiver and Transmitter
UDP	User Datagram Protocol
UI	Un-numbered Information Frame
SABM	Set Asynchronous Balanced Mode Extended Frame
SDL	System Description Language
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SUNSAT	Stellenbosch University Satellite
SSID	Secondary Station Identifier
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
XID	Exchange Identification Frame

Stellenbosch University http://scholar.sun.ac.za

"Not everything that can be counted counts, and not everything that counts can be counted."

- *Albert Einstein*

Stellenbosch University http://scholar.sun.ac.za

Chapter 1

Introduction

1.1 Problem Statement

It is a well-known fact that once a satellite has been launched, it is virtually impossible to have physical contact with it. As a result, all interactions with it will have to take place from a remote location, as can be seen from Figure 1.1.

The communication channel has to make it possible for the ground control station to reliably and efficiently control all subsystems and payloads on the satellite, and also provide the ability to receive, interpret, and execute high-level commands by transforming them into the relevant subsystem or payload commands. The payloads and the subsystems on the satellite will be generating data in response to these commands, and the on-board satellite system has to provide a means of retaining / storing this information until it can be transferred to the ground station [Ste99, p.3].

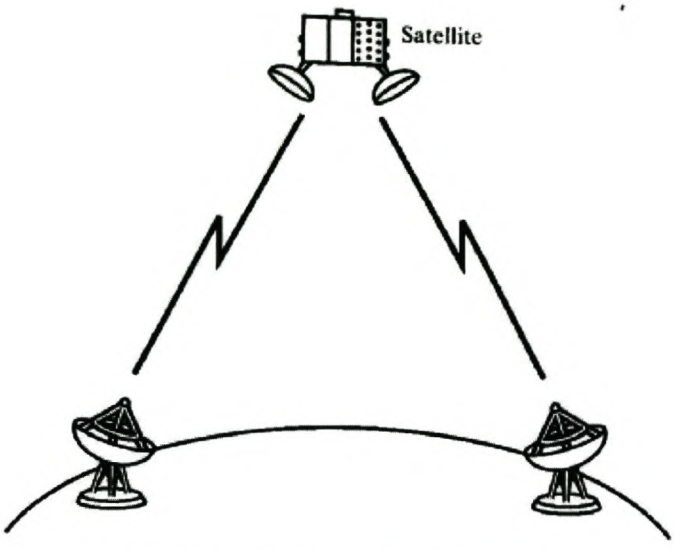
Since a satellite is an expensive remote device and is controlled from the ground, its operation calls for a reliable & efficient way of communication between the two, which will ensure the following:

- The ground station is in total control of the satellite throughout the lifespan of the satellite.

- The satellite correctly receives, interprets and executes all the commands sent by the ground station.

Stellenbosch University http://scholar.sun.ac.za

- Errors are detected when they occur and the system (ground station and satellite) is able to recover from them.



(a) Point-to-point link via satellite microwave

Figure 1.1

One way of achieving the above is to make use of an internationally accepted standard communication protocol. In this way everyone trying to use this protocol will reference the same specifications and thus allowing for duplication & uniformity so that everyone using the protocol can be able to communicate.

The satellite subsystem responsible for achieving that kind of communication is known as the *Communication* subsystem. Table 1.1 shows the different subsystems of a typical satellite and their functions, it also reveals how the *Communication* subsystem relates / fits together with the other subsystems [WL99, p. 303].

Stellenbosch University http://scholar.sun.ac.za

On the satellite, the software involved is the kernel that provides real-time software execution, an interface to the communication hardware and the implementation of the different protocols. At the ground station, the software involved is a collection of programs that enable a user to interact with the satellite-based server [Boo96, p.4].

Subsystem	**Principal Functions**
Attitude Determination and Control System	Provides determination and control of the attitude and orbit position of the spacecraft
Communication	Communicates with ground and possible other spacecraft; spacecraft tracking
Command and Data Handling	Processes and distributes commands; processes, stores and formats data
Power	Generates, stores, regulates and distributes electric power
Thermal	Maintains equipment within allowed temperature ranges
Propulsion	Provides thrust to adjust orbit and attitude
Structure and Mechanisms	Provides support structure, booster adapter, and moving parts

Table 1.1: Summary of General Subsystems of a Satellite

The goal then of the thesis is to use Java & AX.25 protocol to provide a reliable communications platform for a next generation satellite, and also understand the implementation trade-offs for the AX.25 protocol in Java. Reasons for choosing Java as an implementation language and preference of AX.25 protocol over other communication protocols will be dealt with in detail in Chapter 2.

1.2 AX.25 and Other Protocols

All rules, formats, and procedures that the two communicating stations / nodes agree upon are collectively called a *protocol*. In a way, the protocol formalizes the interaction

Stellenbosch University http://scholar.sun.ac.za

between these nodes by standardizing the use of a communication channel [Holz91, p.20].

The protocol, then, can contain agreements on the methods used for [Holz99, p20]:

- Initiation and termination of data exchanges,
- Synchronization of senders and receivers,
- Detection and correction transmission errors, and
- Formation and encoding of data.

The above points will be looked at in detail in Chapter 4, in the context of AX.25.

AX.25 is considered the standard protocol for amateur radio use, and is even recognized by many countries as a legal operation mode. Other standards do exist and are used in some areas of amateur radio, such as TCP/IP. Some networking protocols use packet formats other than AX.25, as well. Often, special packet radio protocols will be encapsulated within AX.25 packet frames to ensure compliance with regulations that require packet radio transmissions to be in the form of AX.25.

According to a web site dedicated to packet radio, packet radio (like any mode in the amateur service) provides groups of amateurs with a way of meeting a primary goal: to improve the radio art. Packet radio was a new mode in the early 80's that many of the outstanding amateur experimenters worked on and developed. The result, ten years later, is something that provides many different operating opportunities. No longer is it just packet radio, but now its applications range across bulletin board systems, chat bridges, networking, emergency communications, satellite operations and more ["Why Packet Radio?"].

Stellenbosch University http://scholar.sun.ac.za

1.3 Work Previously Done on the Subject

To determine implementation success, one has to compare his work to what has already been done and achieved previously on the subject by others. In his work [Boo 96], *J. Boot* implemented the application layer communication protocols and integrated different modules forming the protocol stack that will be used for communication with the SUNSAT micro-satellite.

One of those modules is the AX25 packet-radio link-layer protocol. Although *Boot* did not implement the module himself (he claims the implementation of the AX.25 protocol was the focus of another study described in [Bus 95]), he however adapted & designed interfaces for it to fit the requirements of his work and evaluated the module by measuring some of the performance parameters of the AX.25 protocol. It is these performance parameters that will be compared to the results obtained in this thesis. Details will be given in Chapter 5.

1.4 Thesis Overview

This project forms part of the *Communication* subsystem of the satellite. It involves implementation of a standard communication protocol (AX.25) in Java, for future use on satellite on-board computers, allowing amongst other things, communication with the *Telecommand* subsystem. The thesis also looks at the efficiency evaluation of the implementation, and based on the evaluation results, necessary recommendations for optimization are made. The thesis is structured as follows:

- Chapter 2 will discuss the need for knowledge of networking protocol principles and will identify where the AX.25 protocol fits into the Open Systems Interconnection (OSI) reference model. Factors that necessitated the choice of the AX.25 protocol and

Stellenbosch University http://scholar.sun.ac.za

Java as an implementation language are looked at. A brief history and evolution of AX.25 is also presented.

- Specific details and structure of the complete AX.25 protocol will be presented in Chapter 3. The complete model and packet structure will also be looked into. Essential parts of the protocol to be implemented will also be indicated and motivated as to why they are considered essential.

- Basic objectives of the protocol implementation will be given in Chapter 4. This chapter will look at the implementation of the main parts indicated in Chapter 3. The portions of the protocol implemented here were deemed to be sufficient for normal operation, without compromising the integrity of the protocol.

- Measurements and evaluation of the implemented communication software (AX.25) will be discussed in Chapter 5. A comparison of some of the results obtained and the ones achieved in [Boo 96], will also be looked into.

- Chapter 6 will give conclusions and recommendations, informed mainly by the extent to which basic implementation objectives were met and the results drawn from Chapter 5.

Stellenbosch University http://scholar.sun.ac.za

Chapter 2

Background Studies

This chapter discusses the evolution of packet-radio communication from the early days of the ALOHA system to date. The history provides information about where AX.25 protocol comes from and where packet-radio communication might be headed. A brief description of the communications model and the relationship between OSI reference model & AX.25 protocol is also looked at. Some of the factors that influenced the choice of the AX25 protocol and the implementation language (Java) are also highlighted.

2.1 How the Protocol Evolved

It is well documented that the first computer system to employ radio for its communication facility was the ALOHA system at the University of Hawaii. As a result, the system is often called the ancestor of all packet-broadcasting systems. It first went to the air in 1979 [Tan81, p.273].

The ALOHA system was begun to allow people at the University of Hawaii, who were spread out over seven campuses on four islands, to access the main computer center on Oahu without using telephone lines, which were expensive and unreliable. All communications is from a station to the computer center or from the computer center to a station. There is no station-to-station communication.

Stellenbosch University http://scholar.sun.ac.za

When a packet is received at the computer center, it is processed there and is not retransmitted for all other stations to hear. Since incoming packets are not rebroadcast, a station has no way of knowing whether or not the central site correctly received its transmission. As a result, explicit acknowledgments are needed.

Figure 2.1 below shows the essential elements of the ALOHA system.

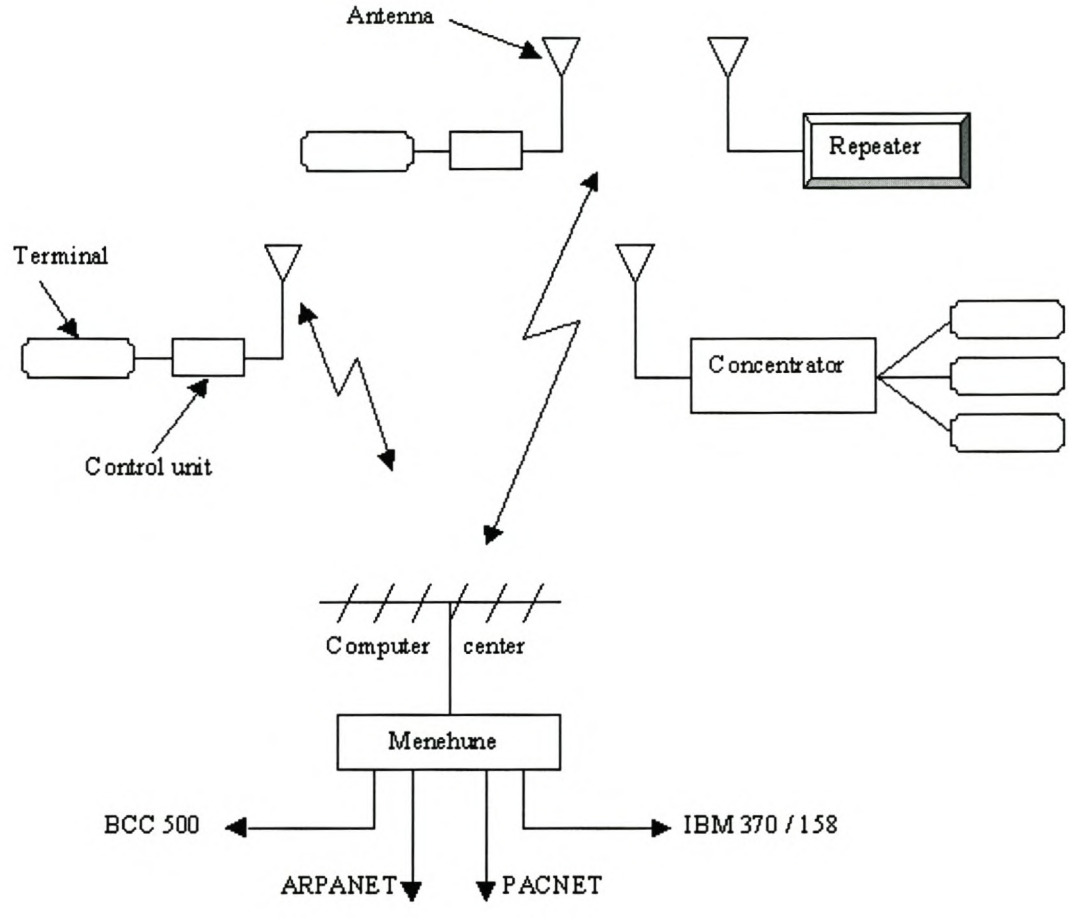


Figure 2.1: The University of Hawaii ALOHA System

Stellenbosch University http://scholar.sun.ac.za

At the central site is a 32K HP 2100 minicomputer, called the Menehune (the Hawaiian word for IMP) that is connected to the antenna. All data in or out of the central site passes through it. The Menehune, in turn, is connected to two large computers, an IBM 370 and the BCC 500, as well as to two other networks, ARPANET and PACNET. Each station has a control unit that buffers some text and handles retransmissions. The original units were hardwired, but later Intel 8080s were used to provide more flexibility. Some stations are connected to concentrators to reduce transmitter/receiver costs.

Packets consist of four parts. First comes a 32-bit header, containing, among other things, the user identification (address) and the packet length. To provide good reliability, the header is followed by a 16-bit checksum. Next comes the data, up to 80 bytes (640 bits), followed by another checksum. The maximum packet is 32 + 16 + 640 + 16 = 704 bits. At 9600 bps, the transmission time for longest packet is 73 msec. When a station has data to send, it proceeds and sends. When the Menehune correctly receives a packet, it inserts an acknowledgement packet into the output stream. If a station does not receive an acknowledgement within a preset time, it assumes that the packet suffered a collision, and retransmits it [Tan81, p.275].

There have been a lot of modifications and improvements to this system (ALOHA), resulting in a number of protocols which evolved from it and AX.25 is one of the latest of these protocols. It is based on the wired network protocol X.25; however, AX.25 was modified to suit the needs of amateur radio. This modification arose from a difference in the transport medium (radios v/s wires) and different addressing schemes.

AX.25 is used between two amateur radio stations in a point-to-point or networked communications environment, and specifies only link layer and physical layer functions. It is not intended to specify any upper-layer protocol other than certain interface requirements to and from other layers.

Stellenbosch University http://scholar.sun.ac.za

2.2 The Need for Background Studies

Since the protocol specifications assume a certain level of understanding of networking protocol principles, a need for background knowledge was identified. This knowledge includes (amongst other things) signal generation, source system, transmission system and the destination system. A simplified communication model is shown in Figure 2.2 below.

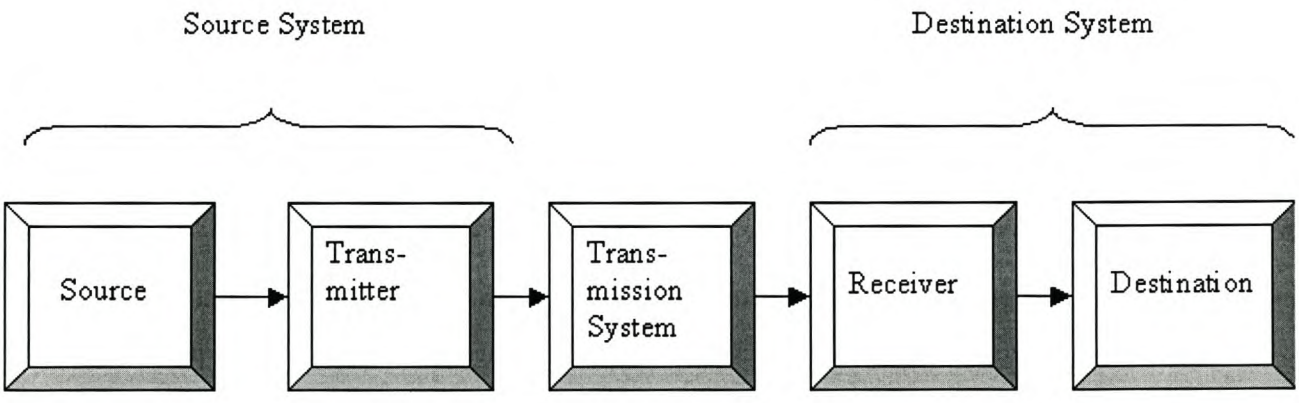


Figure 2.2: Communications Model

The communications model helps with understanding the path a signal takes from the source (sender) to the destination (receiver). This information puts in perspective, the route and the different communication subsystem a packet will traverse from the sender to the receiver.

It was also important that an OSI model (developed by ISO) be thoroughly understood as it serves as a model for computer communications architecture and as a framework for developing protocol standards. Figure 2.3 shows this model.

Stellenbosch University http://scholar.sun.ac.za

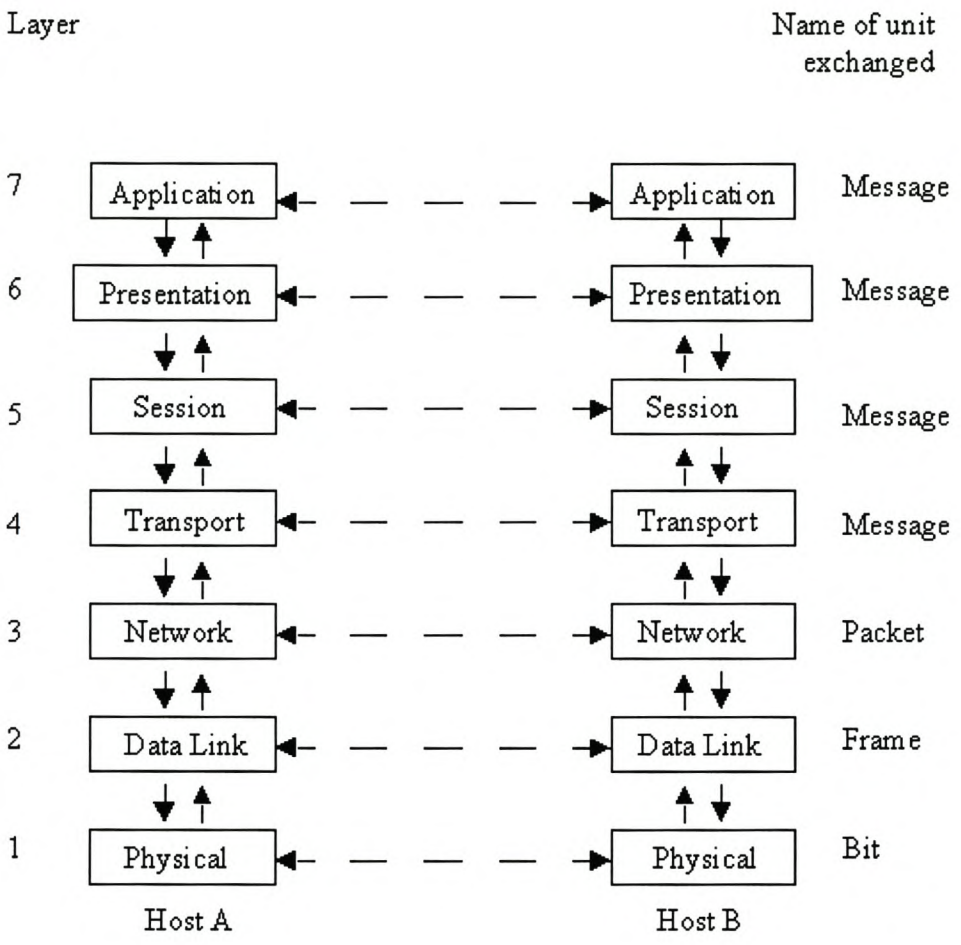


Figure 2.3: The OSI Layers

The principal motivation for the development of the above model was to provide a framework for standardization. Within the model, one or more protocol standards can be developed at each layer. The model defines in general terms the functions to be performed at that layer and facilitates the standards-making process in two ways [Sta00, p.47]:

Stellenbosch University http://scholar.sun.ac.za

- Because the functions of each layer are well defined, standards can be developed independently and simultaneously for each layer. This speeds up the standards-making process.
- Because the boundaries between layers are well defined, changes in standards in one layer need not affect already existing software in another layer. This makes it easier to introduce new standards.

A closer inspection of AX.25 protocol reveals that the protocol provides functionality specified for the following layers of the OSI stack:

- *Data Link Layer* – it conforms to the ISO recommendation for HDLC. The flags, addresses and frame check sequence are part of the AX.25 frame. The protocol provides for frame numbering, flow control, error detection & error control (achieved by retransmission of damaged frames that have not been acknowledged or for which the other station requests a retransmission).
- *Network Layer* – it controls how packets are transferred across the DCE / DCE interface. The protocol allows operation through more than one repeater, creating a routing mechanism for interconnected stations.

2.3 Why AX.25 Protocol

There are a number of features that make AX.25 preferable to other previously used communication protocols. To understand the impact of AX.25 advantages, it is important to know what protocols were in use before. One such protocol was BAUDOT. BAUDOT is a 5-bit code with maximum usable transmission speeds of 75 baud. It offers no error correction and is very slow [Wol95, p.3]. Also, because BAUDOT uses only 5-bit code, only a limited portion of the ASCII character set can be transmitted (primarily restricted to only letters, numbers, and some punctuation).

Stellenbosch University http://scholar.sun.ac.za

The other previously popular protocol used was SITOR. SITOR also uses a 5-bit code, however, it offers two methods of error correction: Forward Error Correction (FEC) and asynchronous hand-shaking (ARQ). According to [ARR93], the error correction offered with SITOR was a significant improvement but the baud rate was still limited to about 50 baud. The AX.25 protocol came as a welcomed solution to the shortcomings of BAUDOT, SITOR, and other earlier protocols. Its advantages include, but are not limited to the following:

- The AX.25 protocol ensures error-free communications.
- A closer look at AX.25 shows that it is a go-back-N protocol (i.e. retransmission of packets due to errors, are done from a specified packet number) – this means the correct packet sequence is maintained, thus minimizing the buffer storage for its implementation.
- It has the ability to transmit an 8-bit code – this makes the sending of all characters of the ASCII table, including non-printable characters, possible. This includes transmission of binary files, executable files (i.e. ending with ".EXE"), graphic images, and digitized audio and video files [Bri94].
- It adds both the senders and receivers' addresses in each frame it sends, in order to conform to FCC rules - it is required by the FCC rules that at the beginning and end of transmission of every thirty minutes of communication, a signaling terminal must broadcast its address [KST, p.3].
- Most link-layer protocols assume one primary (or master) device is connected to one or more secondary (or slave) device(s). This type of unbalanced operation is not practical in a shared-RF amateur radio environment. Instead, AX.25 assumes that both ends are of the same class, thereby eliminating the two different classes of devices [Fox84].

Even when compared to a modern protocol like TCP, AX.25 is still considered superior (for satellite networks) due to TCP's high communications overhead and the TCP

Stellenbosch University http://scholar.sun.ac.za

assumption that every loss of segment encountered is caused by link congestion and subsequently slows down [Sch97, p.4]. (N.B: The assumption is true for copper or fiber based networks, but need not be true for satellite based networks because of their higher error rate).

Finally, one more benefit of the AX.25 protocol is that it has an extensive, proven space history unlike most other protocols. It has been used successfully, as a communication protocol, by a number of satellites including SUNSAT.

2.4 Java Advantages

Since the protocol implementation will be in Java, this section will look at some of the advantages of Java as a programming language. An in-depth discussion of Java advantages is well covered in [Cho], this section will however, only cover those advantages that have a direct impact on the thesis. As described by *Sun Microsystems* in [Hei], Java offers the following advantages for embedded device manufacturers and software developers:

2.4.1 Portability

Platform independence enables code reuse across processors and product lines, allowing device manufacturers to deploy the same applications to a range of target devices. The Java environment itself is portable to new hardware and operating systems, and in fact, the Java compiler itself is written in Java – *the ability to re-use code in large applications like satellite applications is invaluable as it reduces development time & subsequently, costs.*

Stellenbosch University http://scholar.sun.ac.za

2.4.2 Rapid Application Development

Java offers more flexibility during the development cycle. Development can begin on a variety of available desktop environments, well before the targeted deployment hardware is available – *this feature saves time and accelerates software development.*

2.4.3 Java is Simple

Java is object-oriented and object-oriented programming provides greater flexibility, modularity and reusability by modeling the real world (N.B: everything in the world can be modeled as an object which has properties and behaviors). Java has a clean & simple structure that allows programmers to write easy to read and write programs. With embedded Java technology, applications are easy to support & maintain, resulting in greater longevity – *this feature subsequently contributes to the life-span of a satellite. Again in most cases, codes written for satellite applications are huge and can be complex at times, but this feature will help overcome that difficulty by ensuring easy to write & read code.*

2.4.4 Robustness

Java puts a lot of emphasis on early checking for possible errors, as Java compilers are able to detect many problems that would first show up during execution time in other languages. It eliminates certain types of programming constructs in other languages that are prone to errors. For instance, Java does not support direct pointer arithmetic, which eliminates the possibility of overwriting memory and corrupting data. It also has a runtime exception-handling feature to provide programming support for robustness, and can catch & respond to an exceptional situation so that the program can continue its normal execution and terminate gracefully when a runtime error occurs – *this feature enhances reliability of the flight software.*

Stellenbosch University http://scholar.sun.ac.za

2.4.5 Java is Dynamic

Java adapts to an evolving environment, new methods and properties can be added freely in a class without affecting their clients. Also, Java is able to load classes as needed at runtime – *this feature eases the process of updating the flight software of a satellite already in orbit.*

2.4.6 Other Features

Like most other languages, Java supports method inlining – this reduces the dynamic frequency of method invocations, by so doing saving the time needed to perform those method invocations. Other features are its multimedia support (Java supports different image & audio formats); exception–handling capabilities; its strict adherence to class and object-oriented software development; and its automated garbage-collection support – *eliminates many bugs and the risk of memory leaks.*

This chapter looked at the history of packet-radio communication protocols up to AX.25 protocol. Advantages of AX.25 protocol and the Java language were also discussed. Since the project is about implementation of AX.25, the next chapter will discuss the protocol in detail.

Stellenbosch University http://scholar.sun.ac.za

Chapter 3

About The Protocol

The chapter looks at the description of the AX.25 protocol by looking at its model and packet structure. Essential parts of the protocol, which will provide basic objectives of the protocol implementation, will also be indicated.

3.1 Complete Description

As described in [Fox 84], AX.25 protocol provides a mechanism for the reliable transport of data between two signaling terminals. It conforms to the International Standards Organisation (ISO) recommendations 3309, 4335 (including DAD 1&2) and 6256 high-level data link control (HDLC). It also conforms with ANSI X3.66, which describes the advanced data communication control procedure (ADCCP) balanced mode. It follows, in principle, the CCITT X.25 recommendation, with the exception of an extended address field and the addition of the unnumbered information (UI) frame. It also follows the principles of CCITT Recommendation Q.921 (LAPD) in the use of multiple links, distinguished by the address field, on a shared channel.

3.1.1 AX.25 Model

Within the two layers, data link and physical layer, several distinct functional entities can be recognized viz. *Segmenter*, *Data link*, *Management Data link*, *Link Multiplexer*, *Physical* and *Radio*, and these are shown in Figure 3.1. This figure also shows an example of multiple links to the radio port. The link multiplexer described in this standard multiplexes multiple data-link connections into one physical connection.

Stellenbosch University http://scholar.sun.ac.za

A separate data-link machine must be provided for each connection allowed by the implementation.

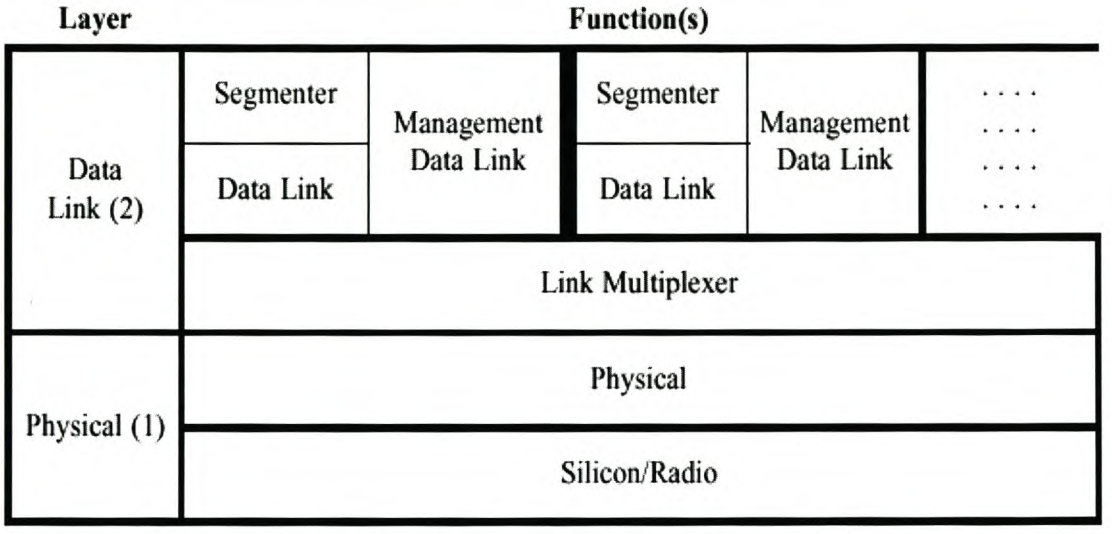


Figure 3.1: AX.25 Finite State Machine Model(multiple stream)

3.1.1.1 Data-Link Service Access Point

The data link layer provides services to Layer 3 through an interface known as the Data-Link Service Access Point (DLSAP), located at the upper boundary of Layer 2 (but not shown in Figure 3.1 above). Associated with each DLSAP is one or more data-link connection endpoint(s) in the data link.

Entities are defined in each layer. Entities may be the Link Multiplexer, Data Link, Management Data Link or Segmenter. Entities in the same layer, but in different systems that must exchange information to achieve a common objective, are called "peer entities." Entities in adjacent layers interact through their common boundary. The services

Stellenbosch University http://scholar.sun.ac.za

provided by the data-link layer are the combination of the services and functions provided by both the data-link layer and the physical layer.

Cooperation between data-link layer entities is governed by a peer-to-peer protocol specific to the layer. For example, when information is to be exchanged between two Layer 3 entities, an association must be established between the entities through the data-link layer using theAX.25 protocol. This association is called a data-link connection. Data-link connections are provided by the data-link layer between two or more DLSAPs.

Layer 3 request services from the data-link layer via command/response interactions known as service "primitives." (Similarly, the interaction between the data-link layer and the physical layer also occurs via service-primitives).

3.1.1.2 Segmenter

The *Segmenter* accepts input from the higher layer through the DLSAP. If the unit of data to be sent exceeds the limits of an AX.25 Information (I) frame or Unnumbered Information (UI) frame, the *Segmenter* breaks that unit of data down into smaller segments for transmission. Incoming segments are reassembled for delivery to the higher layer and passed through the DLSAP. The *Segmenter* passes all other signals unchanged.

3.1.1.3 Data Link

The *Data-link* is regarded as the heart of the AX.25 protocol, since it provides all logic necessary to establish and release connections between two stations and to exchange information in a connectionless (i.e., via UI frames) and connection-oriented (i.e., via I frames with recovery procedures) manner.

Stellenbosch University http://scholar.sun.ac.za

3.1.1.4 Management Data Link

The *Management Data-link* provides for the parameter negotiation of the AX.25 protocol and it provides all logic necessary to negotiate operating parameters between two stations.

3.1.1.5 Link Multiplexer

The *Link Multiplexer* allows one or more data links to share the same physical (radio) channel. The *Link Multiplexer* provides the logic necessary to give each data link an opportunity to use the channel, according to the rotation algorithm embedded within the link multiplexer.

3.1.1.6 Physical

The *Physical* layer can be seen as the interface between the *Data Link* and the radio channel. AX.25 protocol stipulates that the radio channel can operate in either half-duplex or full-duplex mode [Fox84, p.1].

3.1.2 AX.25 Packet Structure

Data is usually sent in small blocks called frames. Each frame consists of several fields that encapsulate the data in its own respective data field. The AX.25 protocol is associated with a specific frame structure.

Stellenbosch University http://scholar.sun.ac.za

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N* 8 Bits	16 Bits	01111110

Figure 3.2: AX.25 Frame Structure

A description of the fields in the AX.25 frame structure follows:

- Flag Field

The flag field is one octet long and is used to delimit the frames. It tells the receiving end when the frame starts and ends and encapsulates the data within. Two frames may share the same flag simply to save space and denote the end of one frame and the beginning of the other frame. This field of the frame construct is unique to all other frames and due to bit stuffing is not allowed to occur anywhere else in the frame [Fox84, p.2].

- Bit Stuffing

In order to ensure that the flag bit sequence doesn't appear accidentally anywhere else in a frame, the sending station monitors the bit sequence for a group of five or more successive "1" bits. Any time five successive "1" bits are sent, the sending station inserts a "0" bit after the fifth "1" bit. During frame reception, any time five successive "1" bits are received, a "0" bit immediately following those five "1" bits is discarded [Fox84, p.4].

- Address Field

The address field identifies the source and destination address for the specific packet. Within the address field, there are several more fields. There are two types of packet radio operation. Either the data will be transmitted from the source to destination directly

Stellenbosch University http://scholar.sun.ac.za

or from source to destination via a repeater, which extends the range of transmission. There are two different implementations of the address field: one for non-repeater (direct) transmission and one for packets that are transmitted through repeaters.

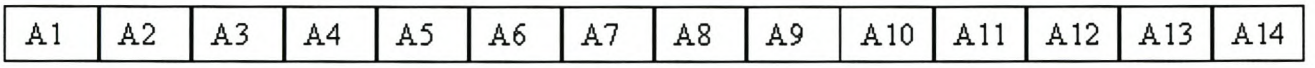


Figure 3.3: General Format for Addressing Field

The entire address field consists of 14 octets. This gives the destination and source fields seven octets each. The destination field is sent first. Sending the destination field first provides the receivers of frames time to check the destination address subfield to see if the frame is addressed to them while the rest of the frame is being received. Figure 3.4 shows a sample of the address field for non-repeater mode of operation.

Stellenbosch University http://scholar.sun.ac.za

OCTET	ASCII	Binary Data	Hex Data
A1	N	10011000	98
A2	J	10010100	94
A3	7	01101110	6E
A4	P	10100000	A0
A5	SPACE	01000000	40
A6	SPACE	01000000	40
A7	SSID	11100000	E0
A8	N	10011000	98
A9	7	01101110	6E
A10	N	10011000	98
A11	E	10001010	8A
A12	M	10011010	9A
A13	SPACE	01000000	40
A14	SSID	CRRSSID0	61

Figure 3.4: Sample Data for a Non-Repeater Addressing Mode

The top octet (A1) is the first one to be sent out. The first bit sent is bit position 0. This bit is the HDLC address extension bit. All of the HDLC bits are set to 0 with the exception of the last bit, which is set to 1. The octet at the end of the source and destinations sections is called the SSID. This feature allows the operator to have more than one packet radio station operating under the same call sign. This is useful when an operator wants to put up a repeater in addition to a regular station, for example.

Stellenbosch University http://scholar.sun.ac.za

The R bits in the SSID octet are reserved for future use and are normally set to 0. The end bit marked with C is the command/response bit. This is used to maintain compatibility with previous versions of the AX.25 protocol. A TNC tests the C bit in the SSID octet of both the destination and source fields. If both bits are set to zero or one then the "distant" device is using the older protocol. If the version is newer, one of the bits is set to one, setting it as a command or response. This helps maintain proper link control during the information transfer state.

The amateur call sign is implemented with seven bit upper-case ASCII characters. They are placed in the left-most bits of the octet to make room for the address extension bit. When a call sign has fewer than six characters, an ASCII space is provided in the field between the last call sign character and the SSID section.

In repeater operations, the C bit in the SSID field is implemented as the repeated bit. When the bit is set to zero it indicates that the frame has not been repeated and vice-versa. It simply indicates whether then frame is from a station or from the output of a repeater. This dictates how the frame is handled upon receipt.

- Control Field

The control field can be implemented with either a three bit or seven bit sequence number. The I frame is used when information is carried. The S frame is a supervisory frame used for connection management. Figure 3.5 shows the basic format of the control field associated with these types of frames.

Stellenbosch University http://scholar.sun.ac.za

Control Field Type	Control-Field Bits							
	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1

Figure 3.5: Seven-Bit Control Field Format

Where:

1. Bit 0 is the first bit sent and bit 7 is the last bit sent of the control field.

2. N(S) is the send sequence number.

3. N(R) is the receive sequence number.

4. The "S" bits are the supervisory function bits, and depending on their encoding they can mean that the station is ready to receive (RR), not ready to receive (RNR) or reject (REJ) frames.

5. The P/F bit is the Poll / Final bit used in all types of frames and in a command (poll) mode to request an immediate reply to a frame.

- Protocol Identifier Field

The PID field appears only in information frames. It identifies whether or not a Layer 3 protocol is in use, or what type of Layer 3 protocol is in use. An Internet Protocol is one of the protocols that can be identified by the PID.

Stellenbosch University http://scholar.sun.ac.za

- Information Field

The Information field of the AX.25 protocol is where the user data is placed, and defaults to a size of 256 octets.

- Frame Check Sequence

The FCS is a 16-bit number calculated by both the sender and the receiver. It ensures that the frame was not corrupted due to noise or other atmospheric interference when sent. The frame ends with the one octet flag that notifies the receiver that the frame is at its end. It is identical to the start flag and can be shared with following frames to reduce data transmission time.

3.2 The Restricted Description

Besides the basic services of accepting and delivering data over a variety of types of communications links, most of the other functionalities found in AX.25 are there to maintain the integrity and robustness of the protocol. As a result, it was decided that in this implementation, some of the funtionalities will be left out and only the most basic and essential be used, without compromising the normal operation of the protocol. The basic objectives, which the protocol implementation is expected to deliver on, will be discussed in Chapter 4.

The use of the following were deemed to be non-essential:

- Multiple Links.
- Multiple Receive Address.
- Response Delay Timer (T2), and Inactive Link Timer (T3).
- Receive Not Ready / Station Busy (RNR).

Stellenbosch University http://scholar.sun.ac.za

- Polling / Final Bit.

Whereas the following were implemented since they provide the basic objectives and maintain the integrity of the protocol:

- Frame Format.
- Link Establishment (SABM).
- Information Transfer (I), with REJ (Reject) and RR (Receive Ready).
- Unnumbered Information Control Field (UI).
- Flow / Window Control (7 packets outstanding).
- Acknowledgement Timer (T1).
- Link Disconnection (DISC).
- Frame Reject Control Field (FRMR).

The AX.25 protocol specification is in Appendix A for detailed description of the above functionalities and others.

This chapter discussed the details of the AX.25 protocol by looking at several distinct functionalities that can be recognized within the protocol. The chapter also discussed essential portions of the protocol that will be implemented. Chapter 4 will look at the protocol implementation and the way it was approached.

Stellenbosch University http://scholar.sun.ac.za

CHAPTER 4

Protocol Implementation

This chapter discusses the software implementation of AX.25 Protocol and its evaluation will be looked at in Chapter 5. Software programs were written in Java. Java provides the ability to write separate software modules with well-defined interfaces. This allows for encapsulation and information hiding which promotes modular development. As a result, different functions can operate at different layers that are independent of each other.

The main points of the protocol design, protocol implementation, and protocol evaluation can be expressed as shown in Table 4.1 [PP80, p.194].

Protocol Realization Steps	**Protocol Realization Phases**
overall design (word definition) specification (formal definition) verification	protocol DESIGN
implementing testing documentation	protocol IMPLEMENTATION
measurement evaluation	protocol EVALUATION

Table 4.1: Main steps of the protocol design, implementation and evaluation.

Stellenbosch University http://scholar.sun.ac.za

Table 4.1 helps with recognizing steps that need to be taken in the design, implementation and finally, evaluation of a protocol. The approach in this thesis will not be different.

The protocol implementation is expected to satisfy the following basic objectives:

- Reliable transmission of data between two signaling terminals
- The ability to recover from the following errors
    1) Reception of out of sequence frames
    2) Reception of incorrect frames – frames with an incorrect *Checksum* or improper addresses
- The ability to initiate or respond appropriately, to a link-resetting procedure after a non-recoverable error has occurred

The testing of the protocol will reveal to what extend the above-mentioned objectives were met.

4.1 Implementation Overview

The decomposition of the communication process into phases provides for differentiating various communication functions performed within the scope of the communication protocol, and in turn the corresponding communication functions maybe assigned to the said individual phases. So we can recognize the line establishment function, the transmission opening function, the data exchange function, the transmission closing function and the line releasing function. However, it is also possible to distinguish several communication functions in one communication phase [PP, p.42].

A thorough study of AX.25 protocol shows that a station having AX.25 protocol running on it might be in one of the following phases:

Stellenbosch University http://scholar.sun.ac.za

- Link-Disconnected Phase,
- Link-Connection Attempt Phase,
- Link-Connection Established Phase,
- Information-Transfer Phase, or
- Link-Disconnection Attempt Phase.

These phases interrelate as shown in Figure 4.1 below and they were all implemented with some restrictions as outlined in section 3.2.

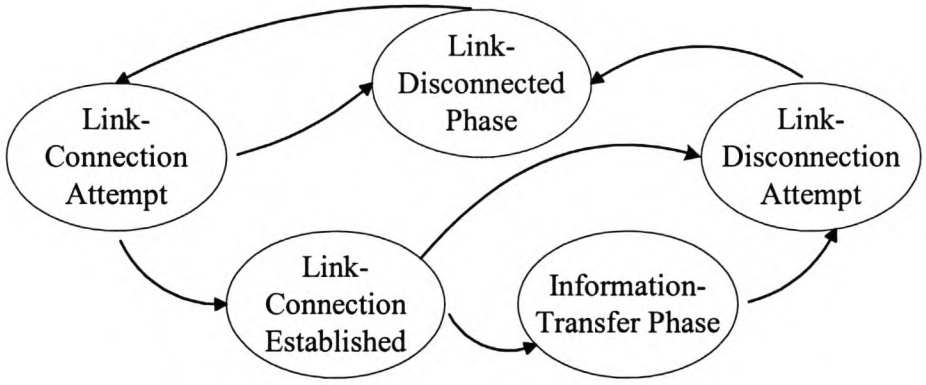


Figure 4.1: Phase-Transition Diagram

It should be noted that the approach of decomposing the communication process into phases (as opposed to the actual implemented states or functions), was chosen so as to simplify the discussion. The details however, of the implemented states & events are clearly captured in the state-tables provided in Appendix B.

The state-tables go as clear as indicating which frames are exchanged when which events occur and what then the next state becomes.

Stellenbosch University http://scholar.sun.ac.za

4.2 Discussion of Communication Phases

4.2.1 Link-Disconnected Phase

When a station is not communicating / connected or is not attempting to setup / accept connection with any other station, then that station is said to be in a *Disconnected Phase.* Figure 4.1 shows that if station A (say) had previously established a connection with another station B, then the only way station A can go back to a *Link-Disconnected Phase* is through a *Link Disconnection Attempt Phase.* Again, a station can choose to remain in a *Link-Disconnected Phase* by denying a *Link-Connection Attempt* by other stations.

4.2.2 Link-Connection Attempt

A station can enter this phase by initiating a *Link-Connection Attempt* (with another station) or just by virtue of receiving a *Link-Connection Attempt* request (from another station). There are two ways to exit the phase, namely:

1. By accepting a *Link-Connection Attempt* request and going to the *Link-Connection Established*, or
2. By denying a *Link-Connection Attempt* request and going to a *Link-Disconnected Phase.*

4.2.3 Link-Connection Established

A station can be in a *Link-Connection Established Phase* by accepting a *Link-Connection Attempt* request. From a *Link-Connection Established Phase*, a station has a choice of entering an *Information-Transfer Phase* or a *Link-Disconnected Attempt phase.*

Stellenbosch University http://scholar.sun.ac.za

4.2.4 Information-Transfer

A station in an *Information-Transfer Phase* is ready and able to receive or send AX.25 data packets with other AX.25 compliant station(s). The *Information-Transfer Phase* is the phase where most of the protocol logic takes place, for example the calculation & verification of the *Checksum* and the identification of *Station Addresses*.

4.2.5 Link-Disconnection Attempt

A station can enter this phase if it has finished with the transmission of data (in the *Information-Transfer Phase*) or maybe while in the *Link-Connection Established Phase*, the station sent or received a *Link-Disconnection* request. A station receiving a *Link-Disconnection* request is obliged to accept the request.

4.3 The Java Class Structure

The Java class structure was organized in a form shown in Figure 4.2. The classes that make up the AX.25 implementation are, *AX.25*, *Data Manager*, *Segmenter*, *ReceiverThread* and *TransmitThread.*

Stellenbosch University http://scholar.sun.ac.za

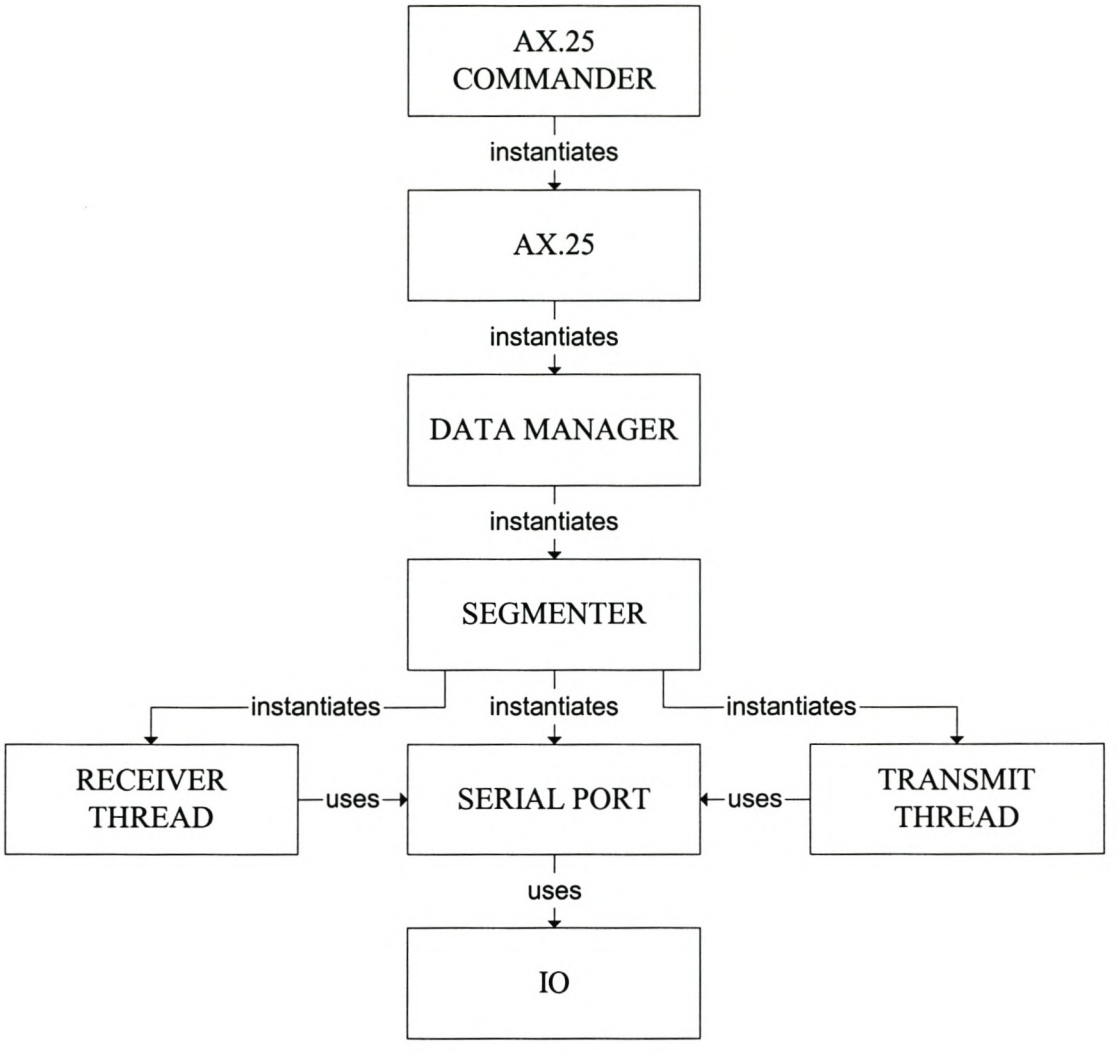


Figure 4.2: Java Class Structure

The *SerialPort* and *IO* classes provide an interface with the computer's serial port whereas *AX.25 Commander* class acts as a main class from where the protocol is being controlled via the usage of the command set discussed in Chapter 5.

Stellenbosch University http://scholar.sun.ac.za

4.4 Per Layer Description of the Implementation

Figure 4.3 below shows different functions (procedures) operating at different layers that are independent of each other. The *Sender* could be the ground station sending Tele-Commands to the *Receiver*, which (in this case) could be the satellite.

A layered discussion of the operation of AX.25 protocol (see Figure 4.3) will reveal how these functions cooperate & complement each other.

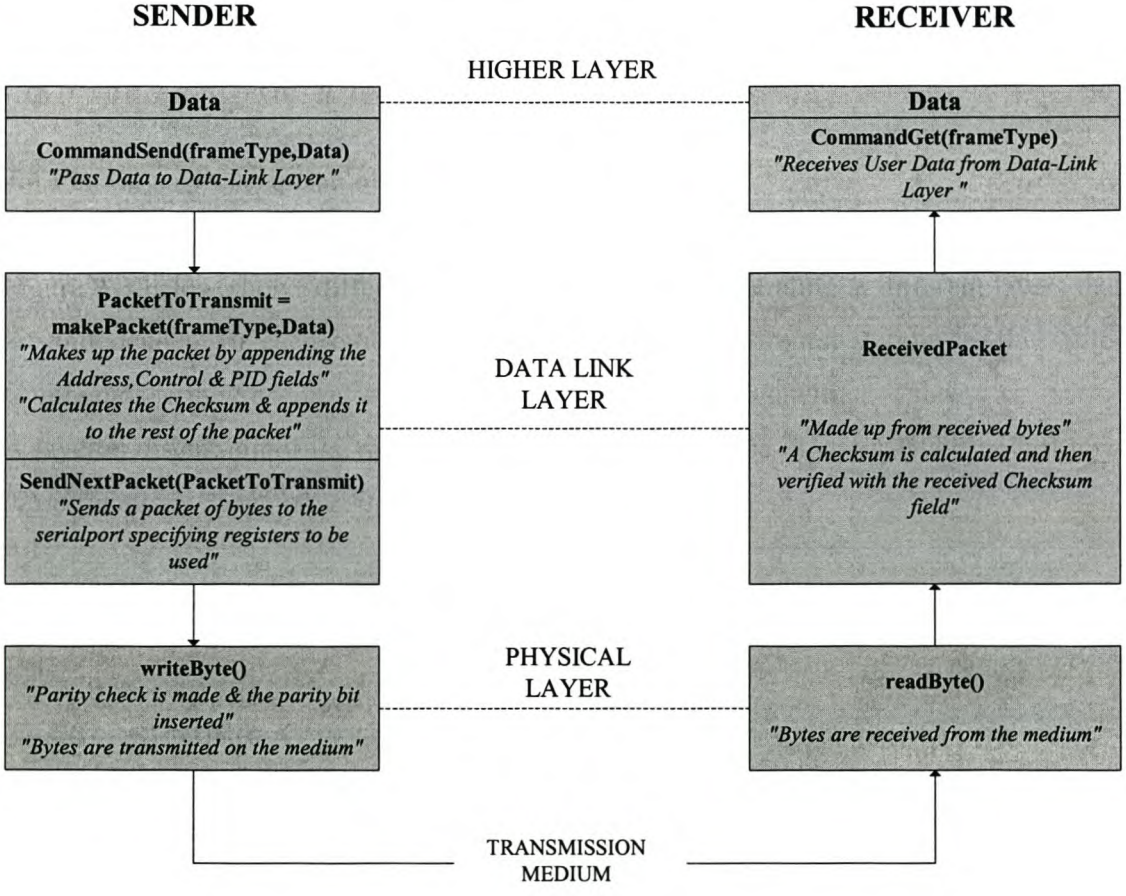


Figure 4.3: Communicating Functions

Stellenbosch University http://scholar.sun.ac.za

4.4.1 Higher Layers

The higher layers (application) pass or receive data to or from the *Data Link Layer.* The data being exchanged might be in different forms, in this implementation character strings were used for simulation purposes.

4.4.2 Data Link Layer Functions (Transmission)

The function ***makePacket(frameType,data)*** makes up the packet by appending the destination & source Addresses, Control & PID fields, then calculates the *Checksum* (FCS) and appends it to the rest of the packet as outlined in Figure 3.2. Where the control field has the sent & received sequence numbers and the PID specifying the kind of layer 3 protocol being used. In this implementation, 11110000 was used for the PID to indicate that there was no layer 3 implemented. Calculation of the *Checksum* was done by XOR-ing the entire fields of the packet.

SendNextPacket(PacketToTransmit) is a function that sends the packet made by ***makePacket(frameType,data)*** for transmission to the serial port, specifying the UART registers to be used. All packets are turned over immediately, meaning therefore that they will be transmitted consecutively on the data link.

4.4.3 Data Link Layer Functions (Reception)

The physical layer passes (frame by frame) all the bytes received from the communications channel (simulated by a serial cable in this implementation) to the data link layer. The incoming bytes are examined, looking for the flag sequence (01111110). The first reception of the flag indicates the end of the previous packet and the beginning of the new packet.

Stellenbosch University http://scholar.sun.ac.za

As the packets are being received, the following checks are performed to verify that the correct packet is being received:

1. The correct address is compared to the address just received.
2. The sequence numbers are compared.
3. The checksum is calculated on the received information bits and compared to the checksum of the sent information bits.

If one of the above checks fails, the packet is classified as invalid and is discarded, then a request for re-transmission of the packet is send to the sender. If the packet meets all the checks, is regarded as valid and all the fields (except the information field) are stripped off the packet.

The higher layer then uses the function *CommandGet(frameType)* to retrieve the received user data.

4.4.4 Physical Layer

Before the actual transmission of data over the communications link can take place, a proper set up has to be established and the processors of the communicating nodes/stations be correctly configured.

4.4.4.1 Port Configuration

Since the protocol implementation test-setup involves two computers connected with a serial cable, communicating via their respective serial ports, a Java class called **serialPort** was written to configure the serial port (i.e. the RS-232 interface) and some of the UART's registers for example the Baud-rate divisor (LSB), Baud-rate divisor (MSB), Line-control register, Modem-control register and the Interrupt-enable register.

Stellenbosch University http://scholar.sun.ac.za

To gain direct access to the input/output devises under Windows, a class **IO** that employs the usage of Native Methods, was used. A brief description of Native Methods, their importance and usage with the Java language, is clearly captured in [Gro98].

Given the implementation test-setup, asynchronous transmission mode was chosen because a user will be keying in, (and subsequently transmitted by a computer), blocks of characters at an indeterminate rate. The following port parameters were used with their values as indicated:

1. Baud-rate = 9600 baud
2. Databits = 8 bits
3. Parity = Odd parity
4. Stop-bits = 2 bits
5. Start-bit = 1 bit

N.B: The usage of start-bit and stop-bits is needed since asynchronous serial data are transmitted & received without a clock or timing signal [Bar00, p.440] (the receiver must be able to re-synchronize at the start of every new character received). The parity bit is used to detect bit errors. The choice of odd or even parity is insignificant since they both achieve the same result, i.e. the detection of 1-bit errors [Fre96, p.127]. The use of 8 data-bits is to take advantage of AX.25's ability to transmit an 8-bit code, as explained in section 2.3. The baud rate should be enough to ensure that the transmitter does not wait on the protocol during transmission – the evaluation results will provide clarity.

4.3.4.2 Interaction with the Data Link Layer

The physical layer uses the two functions ***writeByte( )*** and ***readByte( )***. When the station / node is transmitting, the data link layer uses the function ***SendNextPacket(PacketTotransmit)*** to send packets of bytes to the UART specifying

Stellenbosch University http://scholar.sun.ac.za

registers to be used. Then the physical layer uses the function ***writeByte( )*** to write the bytes to the *Transmitter Holding Register* of the UART.

When the UART has completed transmitting the previous byte or not currently transmitting any data, the contents of the *Transmitter Holding Register* are placed in the *Transmitter Shift Register*. Herein is where the parity check is performed and the parity bit, together with the appropriate stopbits added to the byte, before the entire string of bits is sent out on the serial communication line via the RS-232 interface (as shown in Figure 4.4).

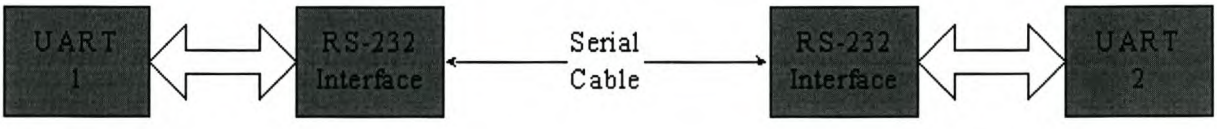


Figure 4.4: The Physical Communication Interface

When the station / node is receiving, the physical layer uses the function ***readByte( )*** to read the contents of the *Receiver Data Register* of the UART and the bytes passed on to the data link layer.

The chapter discussed the Java implementation of the AX.25 protocol, and according to Table 4.1, the next thing to be done is to evaluate the protocol implementation. This will be covered in Chapter 5. Certain performance parameters will be identified, their measurements taken and the analysis of the results presented.

Stellenbosch University http://scholar.sun.ac.za

CHAPTER 5

Evaluation and Results

This chapter discusses protocol evaluation and the method used to achieve it. Protocol evaluation can be done at two levels, namely, operational evaluation (or testing) and performance evaluation. Operational evaluation will look at the proper working of the protocol, i.e. whether the protocol implementation behaves as expected or as outlined by the protocol standard.

Performance evaluation looks at the performance issues deemed to be important, e.g. issues of efficiency of the link and the time taken to complete some of the tasks or functions. The following parameters will be measured:

- The time to service a broadcast packet to be sent,
- The time to decode a broadcast packet,
- The time to set-up a connected-mode link, and
- The time to service a packet to be sent on a link.

Reasons for choosing the above parameters and their measurement results will be presented as well as the analysis of the results.

The main objective of performing an evaluation of the protocol implementation is to identify segments or parts of the protocol that can be optimized. Optimization of the overall performance of the protocol implementation can be achieved via two routes, namely:

- Changing of the implementation design / software structural organization.

Stellenbosch University http://scholar.sun.ac.za

- Investigation & subsequent usage of better performing Java methods instead of the ones currently used, without losing the desired goals of the methods.

5.1 Operational Evaluation

Before the protocol's performance could be evaluated, it is imperative to first test whether the protocol is correctly implemented. For this purpose, an FTP-type program was written to use the protocol implementation.

The main feature of this test program is the command set which is summarized as shown in the table below:

COMMAND	DESCRIPTION
commandConnect <destination_address>	Seeks connection with <destination> address
commandDisconnect	Disconnects from current connection
commandSend <UI,I> <data>	Send data string of type UI or I to the current connection.
commandGet <UI,I>	Check if data is available (and of which kind UI,I or both), and gets it
commandOption <true> / <false>	Sets option to accept connection to True or False
query	Query status of the connection (or AX25 module) at any point of the execution.
quit	Terminates / exits the program, i.e. stops the execution of the AX25 module.

Table 5.1: Command Set of the Protocol Implementation

Stellenbosch University http://scholar.sun.ac.za

In addition to testing for normal operation, an environment that allows errors to be introduced in the communication path had to be created to check how the protocol would recover from those errors. A concept / technique called *Fault Injection* was employed for this purpose. *Fault Injection* is the intentional activation of faults by hardware or software means to be able to observe the system operation under fault conditions [Kop97, p.253]. *Fault Injection* serves two purposes during system evaluation:

- Testing and debugging. During normal operation, faults are events that occur only infrequently, and
- Dependability forecasting.

It is possible to inject faults at the physical level of the hardware (*physical fault injection*) or into the state of the computation (*software implemented fault injection*).

Errors in the transmission medium were simulated by disconnecting the serial cable while packets were in transit. The protocol managed to recover from those errors since it has embedded error-control mechanisms to remedy such errors.

Given the nature of the AX.25 protocol and the implementation, the following errors (in addition to the UART errors, which are, parity error, framing error and overrun error)

- *Frame out-of-sequence error* – when an I frame is received with a correct *Checksum*, but its send sequence number does not match the current receiver's receive-state variable, and
- *Incorrect frame error* – when a frame with an incorrect *Checksum* or an improper address is received

Stellenbosch University http://scholar.sun.ac.za

are expected to occur more, as a result, the operation of the protocol implementation had to be tested under such errors. To achieve this, the following software-implemented fault injections were used for testing and debugging:

- Deliberate use of wrong Frame Variables and Sequence Numbers, thus invoking *Reject* conditions. Used to create *Frame out-of-sequence error.*
- Instead of transmitting a calculated *Checksum* byte, a wrong *Checksum* byte was transmitted thus creating an incorrect Packet. Used to create *Incorrect frame error.*
- Intentional use of incorrect destination addresses. Used to create *Incorrect frame error.*

5.2 Performance Evaluation / Measurements

Measurements were done on both PC's simulating the ground station and the satellite. The PC's were Pentium III, running at 1131 MHz. Measurements were taken using the basic two-channel digital real time oscilloscope (Tektronix TDS 210).

An unused port pin was identified and the Native Method ***writeByte()*** used to write a logical high (1) and a logical low (0) to the pin. The time period it takes a function or code of interest to execute is measured using the above method as shown in the following code extract.

Stellenbosch University http://scholar.sun.ac.za


```
/*-------------------------------------------------------------------------------------------------*/
public void run()
{
  try
  {
    boolean run = true;
    while (run)
    {
      if (DataAvailable)
      {
        pin2 = IO.readByte((short)0x378);                    // Reads the parallel port
        IO.writeByte((short)0x378,(byte) (pin2 | 0x01));     //writes a logical 1 onto the port pin
        sendNextPacket();
        IO.writeByte((short)0x378,(byte) (pin2 & 0xFE));     //writes a logical 0 onto the port pin
      }
    }
  }
  catch(Exception ee){}
}
/*-------------------------------------------------------------------------------------------------*/
```

The time difference is measured at the chosen pin with an oscilloscope probe. The time difference then gives the approximate time period the method ***sendNextPacket()*** takes to execute.

AX.25 protocol is a link-layer protocol that is primarily about the exchange / communication using data packets. A closer inspection of this protocol reveals that the bulk of its logic reside in the data link layer (in relation with the OSI layer stack), and it involves the following functions; acquiring of data (UI or I) to be appended to the packet, the making up of the packet, the calculation & appending of the *Checksum* to the packet, the transmission of the packet to the other communicating node. And on the other node, the functions involve the reception of the packet, its decoding and at times, the building & sending of the response packet.

The time periods the protocol spends on doing the above functions are important, as they will reveal how time is shared among the different functions. The relative time periods will serve as useful information because they will help identify functions that take a lot of

Stellenbosch University http://scholar.sun.ac.za

time to execute (as compared to others) and as a result, those functions can be isolated and optimized as stand-alone functions.

The evaluation of the AX.25 protocol implementation therefore means the examination of the functions described above, as a collective. These functions will be divided into the following parameters to be measured:

- The time to service a broadcast packet to be sent,
- The time to decode a broadcast packet,
- The time to set-up a connected-mode link, and
- The time to service a packet to be sent on a link.

Results obtained in this implementation are compared with the ones achieved in [Boo97, p.91] and the comparisons are given in Table 5.2.

5.3 Results

It should be noted that the protocol implementation supports both C*onnected* and *Connectionless Operation* (*or Packet-Broadcasting*).

Creation of a Packet Transmitted in connectionless mode (256 bytes of data)	**μs**	**% of Total**
Acquisition and addition of Addresses	159.8	66.3
Determination and addition of the Control Field	2.8	1.2
Addition of the Data	55.4	23
Calculation and addition of the Checksum	23	9.5
Total	241	

Stellenbosch University http://scholar.sun.ac.za

The decoding of the packet involves verification of the destination address with the host station's address and the determination of the type of packet received (I, U or S).

Receive Packet in connectionless mode (256 bytes of data)	**µs**	**% of Total**
Decode the AX.25 Packet	176	82.6
Calculation and verification of the Checksum	37.2	17.4
Total	213.2	

To set up a connection, a request is received and a response is sent granting the request.

Receive Connection Request	**µs**	**% of Total**
Decode the Packet received	308	40.4
Calculation and verification of the Checksum	5.2	0.7
Build a response packet (AX.25 AU or DM)	448	58.9
Total	761.2	

In connected mode operation, acknowledgements are received for packets sent and must be sent for packets received, either explicitly or piggybacking on the next packets to be sent.

Receive Packet in connected mode (256 bytes of data)	**µs**	**% of Total**
Decode the Packet received	178.2	41.7
Calculation and verification of the Checksum	37.6	8.8
Build a response packet (AX.25 RR or REJ)	212	49.6
Total	427.8	

Stellenbosch University http://scholar.sun.ac.za

Creation of a Packet Transmitted in connected mode (256 bytes of data)	**μs**	**% of Total**
Acquisition and addition of Addresses	218.1	67.3
Determination and addition of the Control Field	6.6	2
Addition of the Data	75.3	23.2
Calculation and addition of the Checksum	24	7.4
Total	324	

5.4 Analysis of Results

In the analysis of the results, two factors will be looked into, firstly, how the above results relate with each other and secondly, how do they compare with results obtained from a different AX.25 implementation.

The total time taken to process a packet received in connected mode, with 256 bytes of data, is 427.8 μs. Whereas for a packet received in connectionless mode, with 256 bytes of data, the total time is 213.2 μs. The difference (214.6 μs) between these two times can be attributed to fact that packets received in connected mode (i.e. I packets), need to be acknowledged whilst those received in connectionless mode need not be. As a result, 212 μs of that 214.6 μs is spent building the acknowledgement, which leaves 2.6 μs that is used to update receive state variables.

In the creation of a packet transmitted in connected mode, with 256 bytes of data, the total time taken is 324 μs whereas it is 241 μs for a packet transmitted in connectionless mode. The difference (83 μs) between the two times is expected since for an I packet

Stellenbosch University http://scholar.sun.ac.za

transmitted in connected mode, there's additional computation taking place, namely; management of sequence numbers & updating of state variables, determination of the window size with the transmission of every packet and lastly, the protocol implementation runs an additional thread for timing purposes (i.e. for acknowledgement timer T1) when transmitting I packets.

Next, comparative times for two different AX.25 implementations are given in Table 5.2. Column **A** represents results obtained in this implementation with measurements taken using a Pentium III computer running at 1131 MHz, while column **B** represents results obtained in [Boo97, p.91], where measurements were taken using a 386SX computer running at 12 MHz [Boo97, p. 50].

	A (μs)	**B (μs)**
Creation of a packet Transmitted in connectionless mode (256 bytes of data)	241	253
Creation of a packet Transmitted in connected mode (256 bytes of data)	324	724
Decoding of a packet received in connectionless mode (256 bytes of data)	213.2	458
Decoding of a packet received in connected mode (256 bytes of data)	215.8	306
Decoding of a received connection-request packet	313.2	330
Creation of a response (e.g. UA) to a received connection-request packet	448	236
Creation of a response (e.g. RR) to a packet received in connected mode	212	290

Table 5.2: Comparative times for different implementations

Before discussing Table 5.2, it is important to note the following between the two implementations:

- different test setups & measurement techniques were used,
- different computers having different CPU cycles were used,

Stellenbosch University http://scholar.sun.ac.za

- different implementation languages were used (an assumption is made that Modula-2 was used in **B**), and
- **A** only had the AX.25 module running on it while **B** had another extra module, the PACSAT broadcast protocol. These two could have run parallel meaning therefore that they had to share the clock cycles.

The two implementations, as evidenced by Table 5.2, display a similar behaviour that it takes longer to create a packet to be transmitted in connected mode than the one to be transmitted in connectionless mode. However, for the decoding of received packets, the results are not in agreement. It should be stated though that, theoretically, it is expected that it should take a bit of extra time to decode a packet received in connected mode than in connectionless mode due to the added overhead of updating receive state variables for correctly received I packets. This behaviour is displayed only by implementation **A**. The difference though, between **A** and **B**, could be in how that functionality was implemented in both.

Table 5.2 also shows that the time for creation of a response packet to a connection request received, is higher for **A** than for **B**. This can be explained as follows: the main activity in this function is the acquisition of the host station's address and the destination's address from the received request. The function is performed during an establishment of every new connection, meaning therefore that it is for the first time that it gets executed (and this taking place at run time). This means that the Java advantage of method inlining is not helpful at this stage, it only does so after the entire code has at least been executed once, and this exposes the overhead attributed to the presence of the Java Virtual Machine, hence the results.

Now given the repetitive use & non-changing nature of addresses during one connection, method inlining provides the benefit that the method used to acquire the addresses is no

Stellenbosch University http://scholar.sun.ac.za

longer called at the call site hence better times for the creation of other packets of the same length (e.g. RR), exchanged during an already established connection.

Lastly, how does the protocol performance affect the ability of the transmitter to transmit efficiently? An example will be made with a worst case of 324 μs taken to create a packet to be transmitted, and a transmitter with a data rate as high as 1Mbps (N.B: baud rates of this kind are not uncommon in satellite links). It means that it will take the transmitter 2.2ms to transmit a maximum AX.25 packet (276 bytes). It follows therefore that the creation of a packet takes only 15% of the total time to transmit one maximum packet. This allows the protocol implementation with 85% of the time (i.e. 1.87ms) to use to perform other functions including having to handle retransmissions as and when transmission errors occur, before the transmitter is ready to transmit the next packet.

This chapter looked at the evaluation of the protocol implementation. A technique to achieve operational evaluation was discussed and performance results were presented, discussed & compared to results obtained in another implementation. The next chapter will give conclusions and suggest some recommendations.

Stellenbosch University http://scholar.sun.ac.za

Chapter 6

Conclusion and Recommendations

The object of this project was to implement a standard communication protocol (AX.25) in Java, for future use on satellite on-board computers, allowing amongst other things, communication with different subsystems of the satellite. Efficiency evaluation of this Java implementation also formed part of the project.

Background was given on AX.25 protocol and how other special packets radio protocols are encapsulated within AX.25 packet frames to ensure compliance with regulations that require packet radio transmissions to be in the form of AX.25. The history and evolution of packet radio communications from the days of the ALOHA system to AX.25, as it is known today, was also given. Advantages of AX.25 over other communication protocols were also discussed.

A description of the functions that provide the basic objectives and which maintain the integrity of the protocol were identified, and an implementation-structure (the Java class-structure) thereof, formulated. The *design for testability* (the design of a system structure and the provision of mechanisms that facilitate the testing of the system [Kop97, p.252]), among other things, was taken into account in the development of this structure. It was achieved by writing Java classes with well-defined interfaces so that they could be tested in isolation.

The research topic was in part influenced by the rapid emergence of Java as a language of choice, although at the moment most of its usage is found in internet-based applications.

Stellenbosch University http://scholar.sun.ac.za

Even though it is a fact that strongly typed and structured languages (e.g. Modula-2) are a good choice for embedded real-time systems, Java has its own advantages as outlined in Chapter 2.

Java has classes as its fundamental structures and this enhanced the modularity of the protocol implementation. The design identified and subsequently grouped together related functions, in so doing hiding the detail operation of these functions through encapsulation. Modularity is a software-development quality that is both useful and necessary as it provided the protocol implementation with the opportunity to be tested in well-defined software-segments during the development. The inclusion and exclusion of classes from the software during development & evaluation were done with great ease.

Operational evaluation of the protocol was performed by using an FTP-type program, written specifically for this purpose. The program employs the use of a command set which helped in assessing whether the AX.25 module implementation carries out all of its functions and behaves as expected at any given point in its execution. In addition to that, since a notable number of system failures is caused by errors in the fault-management mechanisms [Kop97, p.253], the extensive use of *fault injection* technique for testing and debugging purposes, proved to be invaluable. Operationally, the AX.25 protocol implementation performed satisfactorily as it met all the basic requirements stated in Chapter 4.

However, since the protocol is a go-back-N, it means therefore that some correctly received packets must be retransmitted. This strategy, even though it minimizes buffer storage required for its implementation, is less efficient in its use of the available transmission capacity. There is thus the trade-off between buffer storage requirements and link utilization.

Stellenbosch University http://scholar.sun.ac.za

As expected, the performance evaluation of the AX.25 module reveals that the creation of a packet during information transfer in connected-mode links, uses more resources than using AX.25's connectionless-mode, a result which was also arrived at in [Boo97]; the respective percentages of these two periods being 95% & 44% of what was achieved in [Boo97]. Again, the example given in the analysis of the results also shows that the performance of the protocol implementation ensures that the transmitter does not have to wait on the protocol during transmission, especially when many packets have to be transmitted one after the other - this performance requirement being one of the most important in communication protocol implementations.

Performance results show that on average, the method used in the acquisition & appending of the addresses amounts to 67% of the total time it takes to create a packet. It is recommended that the method be optimized or an alternative method sought. It is also recommended that the Java code be natively compiled to eliminate the overhead presented by the Java Virtual Machine, so as to have an accurate picture of what the speed tradeoffs are.

The data structure commonly used in this implementation is arrays. It is recommended that an investigation into the usage of other data structures like streams be made and the structure yielding better performance be adopted.

The success, ultimately, of a Java version of AX25 module and the general use of the language in satellite systems will depend mainly on two factors, namely: the capabilities or future enhancement of Java as a good real-time language and the future support it will enjoy from the embedded world. Although Java is said to be slow, it should be noted that it is not only about CPU power, but it is also about using functions like garbage collection, method inlining and employing intelligent ways of designing the object-oriented code to enable re-usage of objects. A combination of such advantages will ultimately result in flexible, stable & reliable solution.

Stellenbosch University http://scholar.sun.ac.za

In conclusion, the objective was met in that an AX.25 protocol module, written in Java, was established that provides the required basic functionality. The implementation can still be further optimized for better software structural-organization and for performance in certain areas. This implementation can also serve as a point of departure or reference for further pursuit in writing the AX.25 protocol in Java, for future use on satellite on-board computers.

Stellenbosch University http://scholar.sun.ac.za

BIBLIOGRAPHY

[ARR93] American Radio Relay League. *The ARRL Handbook for Radio Amateurs.* Seventy-first edition, Newington: ARRL, 1993.

[Bar00] Barry B. Brey. *The Intel Microprocessors.* Prentice-Hall International, fifth edition, 2000.

[BNT97] William A. Beech, Douglas E. Nielsen, and Jack Taylor. *Link Access Protocol for Amateur Packet Radio.* Technical report, ARRL, November 1997. Version 2.2.

[Boo96] J. Boot. *Implementation and Integration of Ground and Space Segment Communication Software for the Sunsat Micro-Satellite.* Master's thesis, Department of Electrical and Electronic Engineering, University of Stellenbosch, 1996.

[Bri94] Jack Brindle. *AX.25 Working Papers.* Marietta, GA: packet radio message received, 1994.

[Bus95] JP du Busson. *The Integration of a Link Layer Protocol on a Satellite Subsystem.* Honours in Computer Science, Department of Computer Science, University of Stellenbosch, 1995.

[Cho] Pradnya Choudhari. *Java Advantages and Disadvantages*. Article, ArizonaCommunity.com. http://arizonacommunity.com/articles/java_32001.shtml (October 2002).

Stellenbosch University http://scholar.sun.ac.za

[Fox84] Terry L. Fox. *AX.25 Amateur Packet-Radio Link-Layer Protocol, Version 2.0,* ARRL, October 1984.

[Fre96] Fred Halsall. *Data Communicaitons, Computer Networks and Open Systems.* Addison-Wesley Publishing Company, fourth edition, 1996.

[Gro98] Eugene de Beer. *The Communication of Java with Hardware by means of Native Methods.* B Eng project thesis. Department of Electrical and Electronic Engineering, University of Stellenbosch, 1998.

[Hei] Janice J. Heiss. *Embedded Systems Go Real Time With Java Technology.* http://java.sun.com/features/2001/04/embedded.html (October 2002).

[Holz91] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.

[Kop97] Hermann Kopetz. *Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

[KST] Faiqa Khan, Michael A. Shea and Jamie E. Tamm. *AX.25 Protocol.* CS 460 Final Project. http://pr.erau.edu/~sheam/ (February 2001).

[PP80] J. Puzman and R. Porizek. *Communication Control in Computer Networks.* Wiley, 1980.

[Sch97] Andreas Schilke. *TCP Over Satellite Links.* In Seminar "Broadband Networking Technology", 1997. http://www-tkn.ee.tu-berlin.de/curricula/ss97/bnt97/schilke.html (October 2002).

Stellenbosch University http://scholar.sun.ac.za

[Sta00] William Stallings. *Data and Computer Communications.* Prentice-Hall International, sixth edition, 2000.

[Ste99] N. Steenkamp. *Development of the On Board Computer, Flight Software for SUNSAT 1.* Master's thesis, Department of Electrical and Electronic Engineering, University of Stellenbosch, 1999.

[Tan81] Andrew S. Tanenbaum. *Computer Networks*. Prentice-Hall International, 1981

["Why Packet Radio?"] http://www.tapr.org/tapr/html/Fpktprm1.html (March 2001).

[WL99] James R. Wertz and Wiley J. Larson. *Space Mission Analysis and Design.* Microcosm Press and Kluwer Academic Publishers, third edition, 1999.

[Wol95] Daniel V. Wolff, Jr. *The AX.25 Protocol Used in Radio Networks: A Technical Study and Definition of This Widely Used Data Protocol.* D-54657 Neidenbach, Germany, 1995.

Stellenbosch University http://scholar.sun.ac.za

Appendix A

Version 2.0 October 1984

2. AX.25 Link-Layer Protocol Specification

2.1 Scope and Field of Operation

In order to provide a mechanism for the reliable transport of data between two signaling terminals, it is necessary to define a protocol that can accept and deliver data over a variety of types of communications links. The AX.25 Link-Layer
Protocol is designed to provide this service, independent of any other level that may or may not exist.

This protocol conforms to ISO Recommendations 3309, 4335 (including DAD 1&2) and 6256 high-level data link control (HDLC) and uses some terminology found in these documents. It also conforms with ANSI X3.66, which describes ADCCP, balanced mode.

This protocol follows, in principle, the CCITT X.25 Recommendation, with the exception of an extended address field and the addition of the Unnumbered Information (UI) frame. It also follows the principles of CCITT Recommendation Q.921 (LAPD) in the use of multiple links, distinguished by the address field, on a single shared channel. As defined, this protocol will work equally well in either half- or full-duplex Amateur Radio environments.

This protocol has been designed to work equally well for direct connections between two individual amateur packet-radio stations or an individual station and a multiport controller.

This protocol allows for the establishment of more than one link-layer connection per device, if the device is so capable.

This protocol does not prohibit self-connections. A self-connection is considered to be when a device establishes a link to itself using its own address for both the source and destination of the frame.

Most link-layer protocols assume that one primary (or master) device (generally called a DCE, or data circuit-terminating equipment) is connected to one or more secondary (or slave) device(s) (usually called a DTE, or data terminating equipment). This type of unbalanced operation is not practical in a shared-RF Amateur Radio environment. Instead, AX.25 assumes that both ends of the link are of the same class, thereby eliminating the two different classes of devices. The term DXE is used in this protocol specification to describe the balanced type of device found in amateur packet radio.

Stellenbosch University http://scholar.sun.ac.za

2.2 Frame Structure

Link layer packet radio transmissions are sent in small blocks of data, called frames. Each frame is made up of several smaller groups, called fields. Fig.1 shows the three basic types of frames. Note that the first bit to be transmitted is on the left side.

First
Bit Sent

Flag	Address	Control	FCS	Flag
01111110	112/560 Bits	8 Bits	16 Bits	01111110

Fig. 1A -- U and S frame construction

First
Bit Sent

Flag	Address	Control	PID	Info.	FCS	Flag
01111110	112/560 Bits	8 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Fig. 1B -- Information frame construction

Each field is made up of an integral number of octets (or bytes), and serves a specific function as outlined below.

2.2.1 Flag Field

The flag field is one octet long. Since the flag is used to delimit frames, it occurs at both the beginning and end of each frame. Two frames may share one flag, which would denote the end of the first frame, and the start of the next frame. A flag consists of a zero followed by six ones followed by another zero, or 01111110 (7E hex). As a result of bit stuffing (see 2.2.6, below), this sequence is not allowed to occur anywhere else inside a complete frame.

2.2.2 Address Field

The address field is used to identify both the source of the frame and its destination. In addition, the address field contains the command/response information and facilities for level 2-repeater operation. The encoding of the address field is described in 2.2.13.

2.2.3 Control Field

The control field is used to identify the type of frame being passed and control several attributes of the level 2 connection. It is one octet in length, and its encoding is discussed in 2.3.2.1, below.

2.2.4 PID Field

Stellenbosch University http://scholar.sun.ac.za

The Protocol Identifier (PID) field shall appear in information frames (I and UI) only. It identifies what kind of layer 3 protocol, if any, is in use. The PID itself is not included as part of the octet count of the information field. The encoding of the PID is as follows:

M S B	L S B	
yy01yyyy		AX.25 layer 3 implemented.
yy10yyyy		AX.25 layer 3 implemented.
11001100		Internet Protocol datagram layer 3 implemented.
11001101		Address resolution protocol layer 3 implemented.
11110000		No layer 3 implemented.
11111111		Escape character. Next octet contains more Level 3 protocol information.

Where:
A y indicates all combinations used.
Note:
All forms of yy11yyyy and yy00yyyy other than those listed above are reserved at this time for future level 3 protocols. The assignment of these formats is up to amateur agreement. It is recommended that the creators of level 3 protocols contact the ARRL Ad Hoc Committee on Digital Communications for suggested encodings.

2.2.5 Information Field

The information field is used to convey user data from one end of the link to the other. I fields are allowed in only three types of frames: the I frame, the UI frame, and the FRMR frame. The I field can be up to 256 octets long, and shall contain an integral number of octets. These constraints apply prior to the insertion of zero bits as specified in 2.2.6, below. Any information in the I field shall be passed along the link
transparently, except for the zero-bit insertion (see 2.2.6) necessary to prevent flags from accidentally appearing in the I field.

2.2.6 Bit Stuffing

In order to assure that the flag bit sequence mentioned above doesn't appear accidentally anywhere else in a frame, the sending station shall monitor the bit sequence for a group of five or more contiguous one bits. Any time five contiguous one bits are sent the sending station shall insert a zero bit after the first one bit. During frame reception, any time five contiguous one bits are received, a zero bit immediately following five one bits shall be discarded.

2.2.7 Frame-Check Sequence

The frame-check sequence (FCS) is a sixteen-bit number calculated by both the sender and receiver of a frame. It is used to insure that the frame was not corrupted by the

Stellenbosch University http://scholar.sun.ac.za

medium used to get the frame from the sender to the receiver. It shall be calculated in accordance with ISO 3309 (HDLC) Recommendations.

2.2.8 Order of Bit Transmission

With the exception of the FCS field, all fields of an AX.25 frame shall be sent with each octet's least-significant bit first. The FCS shall be sent most-significant bit first.

2.2.9 Invalid Frames

Any frame consisting of less than 136 bits (including the opening and closing flags), not bounded by opening and closing flags, or not octet aligned (an integral number of octets), shall be considered an invalid frame by the link layer. See also 2.4.4.4, below.

2.2.10 Frame Abort

If a frame must be prematurely aborted, at least fifteen contiguous ones shall be sent with no bit stuffing added.

2.2.11 Interframe Time Fill

Whenever it is necessary for a DXE to keep its transmitter on while not actually sending frames, the time between frames should be filled with contiguous flags.

2.2.12 Link Channel States

Not applicable.

2.2.13 Address-Field Encoding

The address field of all frames shall be encoded with both the destination and source amateur call signs for the frame. Except for the Secondary Station Identifier (SSID), the address field should be made up of upper-case alpha and numeric ASCII characters only. If level 2 amateur "repeaters" are to be used, their call signs shall also be in the address field.

The HDLC address field is extended beyond one octet by assigning the least-significant bit of each octet to be an "extension bit". The extension bit of each octet is set to zero, to indicate the next octet contains more address information, or one, to indicate this is the last octet of the HDLC address field. To make room for this extension bit, the amateur Radio call sign information is shifted one bit left.

2.2.13.1 Nonrepeater Address-Field Encoding

Stellenbosch University http://scholar.sun.ac.za

If level 2 repeaters are not being used, the address field is encoded as shown in Fig. 2. The destination address is the call sign and SSID of the amateur radio station to which the frame is addressed, while the source address contains the amateur call sign and SSID of the station that sent the frame. These call signs are the call signs of the two ends of a level 2 AX.25 link only.

First Octet Sent	Address Field of Frame	
	Destination Address	Source Address
	A1 A2 A3 A4 A5 A6 A7	A8 A9 A10 A11 A12 A13 A14

Fig. 2 -- Nonrepeater Address-Field Encoding

A1 through A14 are the fourteen octets that make up the two address subfields of the address field. The destination subaddress is seven octets long (A1 thru A7), and is sent first. This address sequence provides the receivers of frames time to check the destination address subfield to see if the frame is addressed to them while the rest of the frame is being received. The source address subfield is then sent in octets A8 through A14. Both of these subfields are encoded in the same manner, except that the last octet of the address field has the HDLC address extension bit set.

There is an octet at the end of each address subfield that contains the Secondary Station Identifier (SSID). The SSID subfield allows an Amateur Radio operator to have more than one packet-radio station operating under the same call sign. This is useful when an amateur wants to put up a repeater in addition to a regular station, for example. The C bits (see 2.4.1.2, below) and H bit (see 2.2.13.2, below) are also contained in this octet, along with two bits which are reserved for future use.

Fig. 3A shows a typical AX.25 frame in the nonrepeater mode of operation.

Octet	ASCII	Bin.Data	Hex Data
Flag		01111110	7E
A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	O	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01100100	68
A11	J	10010100	94

Stellenbosch University http://scholar.sun.ac.za

A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100001	61
Control	I	00111110	3E
PID	none	11110000	F0
FCS	part 1	XXXXXXXX	HH
FCS	part 2	XXXXXXXX	HH
Flag		01111110	7E
Bit position		76543210	

Fig. 3A -- Nonrepeater AX.25 frame

The frame shown is an I frame, not going through a level 2 repeater, from WB4JFI (SSID=0) to K8MMO (SSID=0), with no level 3 protocol. The P/F bit is set; the receive sequence number (N(R)) is 1; the send sequence number (N(S)) is 7.

2.2.13.1.1 Destination Subfield Encoding

Fig. 3 shows how an amateur call sign is placed in the destination address subfield, occupying octets A1 thru A7.

Octet	ASCII	Bin.Data	Hex Data
A1	W	10101110	AE
A2	B	10000100	84
A3	4	01101000	68
A4	J	10010100	94
A5	F	10001100	8C
A6	I	10010010	92
A7	SSID	CRRSSID0	
Bit Position →		76543210	

Fig. 3 -- Destination Field Encoding

Where:

1. The top octet (A1) is the first octet sent, with bit 0 of each octet being the first bit sent, and bit 7 being the last bit sent.
2. The first (low-order or bit 0) bit of each octet is the HDLC address extension bit, which is set to zero on all but the last octet in the address field, where it is set to one.
3. The bits marked "r" are reserved bits. They may be used in an agreed-upon manner in individual networks. When not implemented, they should be set to one.
4. The bit marked "C" is used as the command/response bit of an AX.25 frame, as outlined in 2.4.1.2 below.
5. The characters of the call sign should be standard seven-bit ASCII (upper case only) placed in the leftmost seven bits of the octet to make room for the address extension bit. If the call sign contains fewer than six characters, it should be padded with ASCII spaces between the last call sign character and the SSID octet.

Stellenbosch University http://scholar.sun.ac.za

6. The 0000 SSID is reserved for the first personal AX.25 station. This establishes one standard SSID for "normal" stations to use for the first station.

2.2.13.2 Level 2 Repeater-Address Encoding

If a frame is to go through level 2 amateur packet repeater(s), there is an additional address subfield appended to the end of the address field. This additional subfield contains the call sign(s) of the repeater(s) to be used. This allows more than one repeater to share the same RF channel. If this subfield exists, the last octet of the source subfield has its address extension bit set to zero, indicating that more address-field data follows. The repeater-address subfield is encoded in the same manner as the destination and source address subfields, except for the most-significant bit in the last octet, called the "H" bit. The H bit is used to indicate whether a frame has been repeated or not.

In order to provide some method of indicating when a frame has been repeated, the H bit is set to zero on frames going to a repeater. The repeater will set the H bit to one when the frame is retransmitted. Stations should monitor the H bit, and discard any frames going to the repeater (uplink frames), while operating through a repeater. Fig. 4 shows how the repeater-address subfield is encoded. Fig. 4A is an example of a complete frame after being repeated.

Octet	ASCII	Bin.Data	Hex Data
A15	W	10101110	AE
A16	B	10000100	84
A17	4	01101000	68
A18	J	10010100	94
A19	F	10001100	8C
A20	I	10010010	92
A21	SSID	HRRSSID1	
Bit Order →		76543210	

Fig. 4 -- Repeater-address encoding

Where:

1. The top octet is the first octet sent, with bit 0 being sent first and bit 7 sent last of each octet.
2. As with the source and destination address subfields discussed above, bit 0 of each octet is the HDLC address extension bit, which is set to zero on all but the last address octet, where it is set to one.
3. The "R" bits are reserved in the same manner as in the source and destination subfields.
4. The "H" bit is the has-been-repeated bit. It is set to zero whenever a frame has not been repeated, and set to one by the repeater when the frame has been repeated.

Octet	ASCII	Bin.Data	Hex Data
Flag		01111110	7E

Stellenbosch University http://scholar.sun.ac.za

A1	K	10010110	96
A2	8	01110000	70
A3	M	10011010	9A
A4	M	10011010	9A
A5	O	10011110	9E
A6	space	01000000	40
A7	SSID	11100000	E0
A8	W	10101110	AE
A9	B	10000100	84
A10	4	01101000	68
A11	J	10010100	94
A12	F	10001100	8C
A13	I	10010010	92
A14	SSID	01100000	60
A15	W	10101110	AE
A16	B	10000100	84
A17	4	01101000	68
A18	J	10010100	94
A19	F	10001100	8C
A20	I	10010010	92
A21	SSID	11100011	E3
Control	I	00111110	3F
PID	none	11110000	F0
FCS	part 1	XXXXXXXX	HH
FCS	part 2	XXXXXXXX	HH
Flag		01111110	7E
Bit position	→	76543210	

Fig. 4A -- AX.25 frame in repeater mode

The above frame is the same as Fig. 3A, except for the addition of a repeater-address subfield (WB4JFI, SSID=1). The H bit is set, indicating this is from the output of the repeater.

2.2.13.3 Multiple Repeater Operation

The link-layer AX.25 protocol allows operation through more than one repeater, creating a primitive frame routing mechanism. Up to eight repeaters may be used by extending the repeater-address subfield. When there is more than one repeater address, the repeater address immediately following the source address subfield will be considered the address of the first repeater of a multiple-repeater chain. As a frame progresses through a chain of repeaters, each successive repeater will set the H bit (has-been-repeated bit) in its SSID octet, indicating that the frame has been successfully repeated through it. No other changes to the frame are made (except for the necessary recalculation of the FCS). The destination station can determine the route the frame took to each it by examining the address field.

Stellenbosch University http://scholar.sun.ac.za

The number of repeater addresses is variable. All but the last repeater address will have the address extension bits of all octets set to zero, as will all but the last octet (SSID octet) of the last repeater address. The last octet of the last repeater address will have the address extension bit set to one, indicating the end of the address field.
It should be noted that various timers (see 2.4.7, below) may have to be adjusted to accommodate the additional delays encountered when a frame must pass through a multiple-repeater chain, and the return acknowledgement must travel through the same path before reaching the source device. It is anticipated that multiple-repeater operation is a temporary method of interconnecting stations over large distances until such time that a layer 3 protocol is in use. Once this layer 3 protocol becomes operational, repeater chaining should be phased out.

2.3 Elements of Procedure

2.3.1 The elements of procedure are defined in terms of actions that occur on receipt of frames.

2.3.2 Control-Field Formats and State Variables

2.3.2.1 Control-Field Formats

The control field is responsible for identifying the type of frame being sent, and is also used to convey commands and responses from one end of the link to the other in order to maintain proper link control.

The control fields used in AX.25 use the CCITT X.25 control fields for balanced operation (LAPB), with an additional control field taken from ADCCP to allow connectionless and round-table operation. There are three general types of AX.25 frames. They are the Information frame (I frame), the Supervisory frame (S frame), and the Unnumbered frame (U frame). Fig. 5 shows the basic format of the control field associated with these types of frames.

Control-Field	Control-Field Bits							
Type	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	M	M	1	1

Fig. 5 -- Control-field formats

Where:
1. Bit 0 is the first bit sent and bit 7 is the last bit sent of the control field.
2. N(S) is the send sequence number (bit 1 is the LSB).
3. N(R) is the receive sequence number (bit 5 is the LSB).
4. The "S" bits are the supervisory function bits, and their encoding is discussed in 2.3.4.2.

Stellenbosch University http://scholar.sun.ac.za

5. The "M" bits are the unnumbered frame modifier bits and their encoding is discussed in 2.3.4.3.
6. The P/F bit is the Poll/Final bit. Its function is described in 2.3.3. The distinction between command and response, and therefore the distinction between P bit and F bit, is made by addressing rules discussed in 2.4.1.2.

2.3.2.1.1 Information-Transfer Format

All I frames have bit 0 of the control field set to zero. N(S) is the sender's send sequence number (the send sequence number of this frame). N(R) is the sender's receive sequence number (the sequence number of the next expected received frame).
These numbers are described in 2.3.2.4. In addition, the P/F bit is to be used as described in 2.4.2.

2.3.2.1.2 Supervisory Format

Supervisory frames are denoted by having bit 0 of the control field set to one, and bit 1 of the control field set to zero. S frames provide supervisory link control such as acknowledging or requesting retransmission of I frames, and link-level window control. Since S frames do not have an information field, the sender's send variable and the receiver's receive variable are not incremented for S frames. In addition, the P/F bit is used as described in 2.4.2.

2.3.2.1.3 Unnumbered Format

Unnumbered frames are distinguished by having both bits 0 and 1 of the control field set to one. U frames are responsible for maintaining additional control over the link beyond what is accomplished with S frames. They are also responsible for establishing and terminating link connections. U frames also allow for the transmission and reception of information outside of the normal flow control. Some U frames may contain information and PID fields. The P/F bit is used as described in 2.4.2.

2.3.2.2 Control-Field Parameters

2.3.2.3 Sequence Numbers

Every AX.25 I frame shall be assigned, modulo 8, a sequential number from 0 to 7. This will allow up to seven outstanding I frames per level 2 connection at a time.

2.3.2.4 Frame Variables and Sequence Numbers

2.3.2.4.1 Send State Variable V(S)

Stellenbosch University http://scholar.sun.ac.za

The send state variable is a variable that is internal to the DXE and is never sent. It contains the next sequential number to be assigned to the next transmitted I frame. This variable is updated upon the transmission of each I frame.

2.3.2.4.2 Send Sequence Number N(S)

The send sequence number is found in the control field of all I frames. It contains the sequence number of the I frame being sent. Just prior to the transmission of the I frame, N(S) is updated to equal the send state variable.

2.3.2.4.3 Receive State Variable V(R)

The receive state variable is a variable that is internal to the DXE. It contains the sequence number of the next expected received I frame. This variable is updated upon the reception of an error-free I frame whose send sequence number equals the present received state variable value.

2.3.2.4.4 Received Sequence Number N(R)

The received sequence number is in both I and S frames. Prior to sending an I or S frame, this variable is updated to equal that of the received state variable, thus implicitly acknowledging the proper reception of all frames up to and including
N(R) - 1.

2.3.3 Functions of Poll/Final (P/F) Bit

The P/F bit is used in all types of frames. It is used in a command (poll) mode to request an immediate reply to a frame. The reply to this poll is indicated by setting the response (final) bit in the appropriate frame. Only one outstanding poll condition per direction is allowed at a time. The procedure for P/F bit operation is described in 2.4.2.

2.3.4 Control Field Coding for Commands and Responses

The following commands and responses, indicated by their control field encoding, are to be use by the DXE:

2.3.4.1 Information Command Frame Control Field

The function of the information (I) command is to transfer across a data link sequentially numbered frames containing an information field. The information-frame control field is encoded as shown in Fig. 6. These frames are sequentially numbered by the N(S) subfield to maintain control of their passage over the link-layer connection.

Stellenbosch University http://scholar.sun.ac.za

Control Field Bits							
7	6	5	4	3	2	1	0
N(R)		P	N(S)				0

Fig. 6 -- I frame control field

2.3.4.2 Supervisory Frame Control Field

The supervisory frame control fields are encoded as shown in Fig. 7.

Control Field Bits	7	6	5	4	3	2	1	0
Receive Ready RR		N(R)		P/F	0	0	0	1
Receive Not Ready RNR		N(R)		P/F	0	1	0	1
Reject REJ		N(R)		P/F	1	0	0	1

Fig. 7 -- S frame control fields

The Frame identifiers:

C or SABM	Layer 2 Connect Request
D or DISC	Layer 2 Disconnect Request
I	Information Frame
RR	Receive Ready. System Ready To Receive
RNR or NR	Receive Not Ready. TNC Buffer Full
RJ or REJ	Reject Frame. Out of Sequence or Duplicate
FRMR	Frame Reject. Fatal Error
UI	Unnumbered Information Frame. "Unproto"
DM	Disconnect Mode. System Busy or Disconnected.

2.3.4.2.1 Receive Ready (RR) Command and Response

Receive Ready is used to do the following:
1. to indicate that the sender of the RR is now able to receive more I frames.
2. to acknowledge properly received I frames up to, and including N(R)-1, and
3. to clear a previously set busy condition created by an RNR command having been sent.
The status of the DXE at the other end of the link can be requested by sending a RR command frame with the P-bit set to one.

2.3.4.2.2 Receive Not Ready (RNR) Command and Response

Receive Not Ready is used to indicate to the sender of I frames that the receiving DXE is temporarily busy and cannot accept any more I frames. Frames up to N(R)-1 are acknowledged. Any I frames numbered N(R) and higher that might have been caught between states and not acknowledged when the RNR command was sent are not acknowledged. The RNR condition can be cleared by the sending of a UA, RR, REJ, or SABM frame. The status of the DXE at the other end of the link can be requested by sending a RNR command frame with the P bit set to one.

Stellenbosch University http://scholar.sun.ac.za

2.3.4.2.3 Reject (REJ) Command and Response

The reject frame is used to request retransmission of I frames starting with N(R). Any frames that were sent with a sequence number of N(R)-1 or less are acknowledged. Additional I frames may be appended to the retransmission of the N(R) frame if there are any. Only one reject frame condition is allowed in each direction at a time. The reject condition is cleared by the proper reception of I frames up to the I frame that caused the reject condition to be initiated. The status of the DXE at the other end of the link can be requested by sending a REJ command frame with the P bit set to one.

2.3.4.3 Unnumbered Frame Control Fields

Unnumbered frame control fields are either commands or responses. Fig. 8 shows the layout of U frames implemented within this protocol.

Control Field	Type	Control Field Bits
		7 6 5 4 3 2 1 0
Set Asynchronous Balanced Mode-SABM	Cmd	0 0 1 P 1 1 1 1
Disconnect-DISC	Cmd	0 1 0 P 0 0 1 1
Disconnected Mode-DM	Res	0 0 0 F 1 1 1 1
Unnumbered Acknowledge-UA	Res	0 1 1 F 0 0 1 1
Frame Reject-FRMR	Res	1 0 0 F 0 1 1 1
Unnumbered Information-UI	Either	0 0 0 P/F0 0 1 1

Fig. 8 -- U frame control fields

2.3.4.3.1 Set Asynchronous Balanced Mode (SABM) Command

The SABM command is used to place 2 DXEs in the asynchronous balanced mode. This is a balanced mode of operation known as LAPB where both devices are treated as equals. Information fields are not allowed in SABM commands. Any outstanding I frames left when the SABM command is issued will remain unacknowledged. The DXE confirms reception and acceptance of a SABM command by sending a UA response frame at the earliest opportunity. If the DXE is not capable of accepting a SABM command, it should respond with a DM frame if possible.

2.3.4.3.2 Disconnect (DISC) Command

The DISC command is used to terminate a link session between two stations. No information field is permitted in a DISC command frame. Prior to acting on the DISC frame, the receiving DXE confirms acceptance of the DISC by issuing a UA response frame at its earliest opportunity. The DXE sending the DISC enters the disconnected state when it receives the UA response. Any unacknowledged I frames left when this command is acted upon will remain unacknowledged.

2.3.4.3.3 Frame Reject (FRMR) Response

Stellenbosch University http://scholar.sun.ac.za

2.3.4.3.3.1

The FRMR response frame is sent to report that the receiver of a frame cannot successfully process that frame and that the error condition is not correctable by sending the offending frame again. Typically this condition will appear when a frame without an FCS error has been received with one of the following conditions:

1. The reception of an invalid or not implemented command or response frame.
2. The reception of an I frame whose information field exceeds the agreed-upon length. (See 2.4.7.3, below.)
3. The reception of an improper N(R). This usually happens when the N(R) frame has already been sent and acknowledged, or when N(R) is out of sequence with what was expected.
4. The reception of a frame with an information field where one is not allowed, or the reception of a U or S frame whose length is incorrect. Bits W and Y described in 2.3.4.3.3.2 should both be set to one to indicate this condition.
5. The reception of a supervisory frame with the F bit set to one, except during a timer recovery condition (see 2.4.4.9), or except as a reply to a command frame sent with the P bit set to one. Bit W (described in 2.3.4.3.3.2) should be set to one.
6. The reception of an unexpected UA or DM response frame. Bit W should be set to one.
7. The reception of a frame with an invalid N(S). Bit W should be set to one.

An invalid N(R) is defined as one which points to an I frame that previously has been transmitted and acknowledged, or an I frame which has not been transmitted and is not the next sequential I frame pending transmission. An invalid N(S) is defined as an N(S) that is equal to the last transmitted N(R)+k and is equal to the received state variable V(R), where k is the maximum number of outstanding information frames as defined in 2.4.7.4 below.

An invalid or not implemented command or response is
defined as a frame with a control field that is unknown to the receiver of this frame.

2.3.4.3.3.2

When a FRMR frame is sent, an information field is added to the frame that contains additional information indicating where the problem occurred. This information field is three octets long and is shown in Fig. 9.

Information Field Bits

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	Z	Y	X	W	V(R)			C	V(S)			0	Rejected Frame Control Field									
															R										

Fig. 9 -- FRMR frame information field

Where:

Stellenbosch University http://scholar.sun.ac.za

1. The rejected frame control field carries the control field of the frame that caused the reject condition. It is in bits 0-7 of the information field.
2. V(S) is the current send state variable of the device reporting the rejection (bit 9 is the low bit).
3. The CR bit is set to zero to indicate the rejected frame was a command, or one if it was a response.
4. V(R) is the current receive state variable of the device reporting rejection (bit 13 is the low bit).
5. If W is set to 1, the control field received was invalid or not implemented.
6. If X is set to 1, the frame that caused the reject condition was considered invalid because it was a U or S frame that had an information field that is not allowed. Bit W must be set to 1 in addition to the X bit.
7. If Y is set to 1, the control field received and returned in bits exceeded the maximum allowed under this recommendation in 2.4.7.3, below.
8. If A is set to 1, the control field received and returned in bits 1 to 8 contained an invalid N(R).
9. Bits 8, and 20 to 23 are set to 0.

2.3.4.3.4 Unnumbered Acknowledge (UA) Response

The UA response frame is sent to acknowledge the reception and acceptance of a SABM or DISC command frame. A received command is not actually processed until the UA response frame is sent. Information fields are not permitted in a UA frame.

2.3.4.3.5 Disconnected Mode (DM) Response

The disconnected mode response is sent whenever a DXE receives a frame other than a SABM or UI frame while in a disconnected mode. It is also sent to request a set mode command, or to indicate it cannot accept a connection at the moment. The DM response does not have an information field. Whenever a SABM frame is a received, and it is determined that a connection is not possible, a DM frame shall be sent. This will indicate that the called station cannot accept a connection at that time. While a DXE is in the disconnected mode, it will respond to any command other than a SABM or UI frame with a DM response with the P/F bit set to 1.

2.3.4.3.6 Unnumbered Information (UI) Frame

The Unnumbered Information frame contains PID and information fields and is used to pass information along the link outside the normal information controls. This allows information fields to go back and forth on the link bypassing flow control.
Since these frames are not acknowledgeable, if one gets obliterated, there is no way to recover it. A received UI frame with the P bit set shall cause a response to be transmitted. This response shall be a DM frame when in the disconnected state or a RR (or RNR, if appropriate) frame in the information transfer state.

Stellenbosch University http://scholar.sun.ac.za

2.3.5 Link Error Reporting and Recovery

There are several link-layer errors that are recoverable without terminating the connection. These error situations may occur as a result of malfunctions within the DXE, or if transmission errors occur.

2.3.5.1 DXE Busy Condition

When a DXE becomes temporarily unable to receive I frames, such as when receive buffers are full, it will send a Receive Not Ready (RNR) frame. This informs the other DXE that this DXE cannot handle any more I frames at the moment. This condition is usually cleared by the sending of a UA, RR, REJ, or SABM command frame.

2.3.5.2 Send Sequence Number Error

If the send sequence number, N(S), of an otherwise error-free received frame does not match the receive state variable, V(R), a send sequence error has occurred, and the information field will be discarded. The receiver will not acknowledge this frame, or any other I frames, until N(S) matches V(R).

The control field of the erroneous I frame(s) will be accepted so that link supervisory functions such as checking the P/F bit can still be performed. Because of this updating, the retransmitted I frame may have an updated P bit and N(R).

2.3.5.3 Reject (REJ) Recovery

REJ is used to request a retransmission of I frames following the detection of a N(S) sequence error. Only one outstanding "sent REJ" condition is allowed at a time. This condition is cleared when the requested I frame has been received. A DXE receiving the REJ command will clear the condition by resending all outstanding I frames (up to the window size), starting with the one indicated in N(R) of the REJ frame.

2.3.5.4 Time-out Error Recovery

2.3.5.4.1 T1 Timer Recovery

If a DXE, due to a transmission error, does not receive (or receives and discards) a single I frame or the last I frame in a sequence of I frames, it will not detect a send sequence-number error, and therefore will not transmit a REJ. The DXE which transmitted the unacknowledged I frame(s) shall, following the completion of time out period T1, take appropriate recovery action to determine when I frame retransmission should begin as described in 2.4.4.9, below. This condition is cleared by the reception of an acknowledgement for the sent frame(s), or by the link being reset. See 2.4.6.

2.3.5.4.2 Timer T3 Recovery

Stellenbosch University http://scholar.sun.ac.za

Timer T3 is used to assure the link is still functional during periods of low information transfer. Whenever T1 is not running (no outstanding I frames), T3 is used to periodically poll the other DXE of a link. When T3 times out, a RR or RNR
frame is transmitted as a command and with the P bit set. The waiting acknowledgement procedure (2.4.4.9, below) is then executed.

2.3.5.5 Invalid Frame or FCS Error

If an invalid frame is received, or a frame is received with an FCS error, that frame will be discarded with no action taken.

2.3.5.6 Frame Rejection Condition

A frame rejection condition occurs when an otherwise error-free frame has been received with one of the conditions listed in 2.3.4.3.3 above. Once a rejection error occurs, no more I frames are accepted (except for the examination of the P/F bit) until the error is resolved. The error condition is reported to the other DXE by sending a FRMR response frame. See 2.4.5.

2.4 Description of AX.25 Procedures

The following describes the procedures used to setup, use, and disconnect a balanced link between two DXE stations.

2.4.1 Address Field Operation

2.4.1.1 Address Information

All transmitted frames shall have address fields conforming to 2.2.13, above. All frames shall have both the destination device and the source device addresses in the address field, with the destination address coming first. This allows many links to share the same RF channel. The destination address is always the address of the station(s) to receive the frame, while the source address contains the address of the device that sent the frame. The destination address can be a group name or club call sign if the point-to-multipoint operation is allowed. Operation with destination addresses other than actual amateur call signs is a subject for further study.

2.4.1.2 Command/Response Procedure

AX.25 Version 2.0 has implemented the command/response information in the address field. In order to maintain compatibility with previous versions of AX.25, the command/response information is conveyed using two bits. An upward-compatible AX.25 DXE can determine whether it is communicating with a DXE using an older version of this protocol by testing the command/response bit information located in bit 7 of the SSID octets of both the destination and source address subfields. If both C bits are

Stellenbosch University http://scholar.sun.ac.za

set to zero, the device is using the older protocol. The newer version of the protocol always has one of these two bits set to one and the other set to zero, depending on whether the frame is a command or a response. The command/response information is encoded into the address field as shown in Fig. 10.

Frame Type	Dest. SSID C-Bit	Source SSID C-Bit
Previous versions	0	0
Command (V.2.0)	1	0
Response (V.2.0)	0	1
Previous versions	1	1

Fig. 10 -- Command/Response encoding

Since all frames are considered either commands or responses, a device shall always have one of the bits set to one, and the other bit set to zero. The use of the command/response information in AX.25 allows S frames to be either commands or responses. This aids maintenance of proper control over the link during the information transfer state.

2.4.2 P/F Bit Procedures

The next response frame returned by the DXE to a SABM or DISC command with the P bit set to 1 will be a UA or DM response with the F bit set to 1. The next response frame returned to an I frame with the P bit set to 1, received during the information transfer state, will be a RR, RNR, or REJ response with the F bit set to 1. The next response frame returned to a supervisory command frame with the P bit set to 1, received during the information transfer state, will be a RR, RNR, or REJ response frame with the F bit set to 1. The next response frame returned to a S or I command
frame with the P bit set to 1, received in the disconnected state, will be a DM response frame with the F bit set to 1. The P bit is used in conjunction with the time-out recovery condition discussed in 2.3.5.4, above. When not used, the P/F bit is set to zero.

2.4.3 Procedures For Link Set-Up and Disconnection

2.4.3.1 LAPB Link Connection Establishment

When one DXE wishes to connect to another DXE, it will send a SABM command frame to that device and start timer (T1). If the other DXE is there and able to connect, it will respond with a UA response frame, and reset both of its internal state
variables (V(S) and V(R)). The reception of the UA response frame at the other end will cause the DXE requesting the connection to cancel the T1 timer and set its internal state variables to 0. If the other DXE doesn't respond before T1 times out, the
device requesting the connection will re-send the SABM frame, and start T1 running again. The DXE should continue to try to establish a connection until it has tried unsuccessfully N2 times. N2 is defined in 2.4.7.2, below. If, upon reception of a SABM

Stellenbosch University http://scholar.sun.ac.za

command, the DXE decides that it cannot enter the indicated state, it should send a DM frame. When receiving a DM response, the DXE sending the SABM
should cancel its T1 timer, and not enter the information-transfer state. The DXE sending a SABM command will ignore and discard any frames except SABM, DISC, UA, and DM frames from the other DXE. Frames other than UA and DM in response to a received SABM will be sent only after the link is set up and if no outstanding SABM exists.

2.4.3.2 Information-Transfer Phase

After establishing a link connection, the DXE will enter the information-transfer state. In this state, the DXE will accept and transmit I and S frames according to the procedure outlined in 2.4.4, below. When receiving a SABM command while in the information-transfer state, the DXE will follow the resetting procedure outlined in 2.4.6 below.

2.4.3.3 Link Disconnection

2.4.3.3.1

While in the information-transfer state, either DXE may indicate a request to disconnect the link by transmitting a DISC command frame and starting timer T1 (see 2.4.7).

2.4.3.3.2

A DXE, upon receiving a valid DISC command, shall send a UA response frame and enter the disconnected state. A DXE, upon receiving a UA or DM response to a sent DISC command, shall cancel timer T1, and enter the disconnected state.

2.4.3.3.3

If a UA or DM response is not correctly received before T1 times out, the DISC frame should be sent again and T1 restarted. If this happens N2 times, the DXE should enter the disconnected state.

2.4.3.4 Disconnected State

2.4.3.4.1

A DXE in the disconnected state shall monitor received commands and react upon the reception of a SABM as described in 2.4.3.1 above and will transmit a DM frame in response to a DISC command.

2.4.3.4.2

Stellenbosch University http://scholar.sun.ac.za

In the disconnected state, a DXE may initiate a link set-up as outlined in connection establishment above (2.4.3.1). It may also respond to the reception of a SABM and establish a connection, or it may ignore the SABM and send a DM instead.

2.4.3.4.3

Any DXE receiving a command frame other than a SABM or UI frame with the P bit set to one should respond with a DM frame with the F bit set to one. The offending frame should be ignored.

2.4.3.4.4

When the DXE enters the disconnected state after an error condition or if an internal error has resulted in the DXE being in the disconnected state, the DXE should indicate this by sending a DM response rather than a DISC frame and follow the link disconnection procedure outlined in 2.4.3.3.3, above. The DXE may then try to re establish the link using the link set-up procedure outlined in 2.4.3.1, above.

2.4.3.5 Collision Recovery

2.4.3.5.1 Collisions in a Half-Duplex Environment

Collisions of frames in a half-duplex environment are taken care of by the retry nature of the T1 timer and retransmission count variable. No other special action needs to be taken.

2.4.3.5.2 Collisions of Unnumbered Commands

If sent and received SABM or DISC command frames are the same, both DXEs should send a UA response at the earliest opportunity, and both devices should enter the indicated state. If sent and received SABM or DISC commands are different, both DXEs should enter the disconnected state and transmit a DM frame at the earliest opportunity.

2.4.3.5.3 Collision of a DM with a SABM or DISC

When an unsolicited DM response frame is sent, a collision between it and a SABM or DISC may occur. In order to prevent this DM from being misinterpreted, all unsolicited DM frames should be transmitted with the F bit set to zero. All SABM and DISC frames should be sent with the P bit set to one. This will prevent any confusion when a DM frame is received.

2.4.3.6 Connectionless Operation

In Amateur Radio, there is an additional type of operation that is not feasible using level 2 connections. This operation is the round table, where several amateurs may be engaged in one conversation. This type of operation cannot be accommodated by AX.25 link-layer

Stellenbosch University http://scholar.sun.ac.za

connections. The way round-table activity is implemented is technically outside the AX.25 connection, but still using the AX.25 frame structure. AX.25 uses a special frame for this operation, called the Unnumbered Information (UI) frame. When this type of operation is used, the destination address should have a code word installed in it to prevent the users of that particular round table from seeing all frames going through the shared RF medium. An example of this is if a group of amateurs are in a round-table discussion about packet radio, they could put PACKET in the destination address, so they would receive frames only from others in the same discussion. An added advantage of the use of AX.25 in this manner is that the source of each frame is in the source address subfield, so software could be written to automatically display who is making what comments. Since this mode is connectionless, there will be no requests for retransmissions of bad frames. Collisions will also occur, with the potential of losing the frames that collided.

2.4.4 Procedures for Information Transfer

Once a connection has been established, as outlined above, both devices are able to accept I, S, and U frames.

2.4.4.1 Sending I Frames

Whenever a DXE has an I frame to transmit, it will send the I frame with N(S) of the control field equal to its current send state variable V(S). Once the I frame is sent, the send state variable is incremented by one. If timer T1 is not running, it should be started. If timer T1 is running, it should be restarted. The DXE should not transmit any more I frames if its send state variable equals the last received N(R) from the other side of the link plus seven. If it were to send more I frames, the flow control window would be exceed, and errors could result. If a DXE is in a busy condition, it may still send I frames as long as the other device is not also busy. If a DXE is in the frame-rejection mode, it will stop sending I frames.

2.4.4.2 Receiving I Frames

2.4.4.2.1

If a DXE receives a valid I frame (one with a correct FCS and whose send sequence number equals the receiver's receive state variable) and is not in the busy condition, it will accept the received I frame, increment its receive state variable, and act in one of the following manners:

1. If it has an I frame to send, that I frame may be sent with the transmitted N(R) equal to its receive state variable V(R) (thus acknowledging the received frame). Alternately, the device may send a RR frame with N(R) equal to V(R), and then send the I frame.

Stellenbosch University http://scholar.sun.ac.za

2. If there are no outstanding I frames, the receiving device will send a RR frame with N(R) equal to V(R). The receiving DXE may wait a small period of time before sending the RR frame to be sure additional I frames are not being transmitted.

2.4.4.2.2

If the DXE is in a busy condition, it may ignore any received I frames without reporting this condition other than repeating the indication of the busy condition. If a busy condition exists, the DXE receiving the busy condition indication should poll the sender of the busy indication periodically until the busy condition disappears. A DXE may poll the busy DXE periodically with RR or RNR frames with the P bit set to one. The reception of I frames that contain zero-length information fields shall be reported to the next level but no information field will be transferred.

2.4.4.3 Reception of Out of Sequence Frames

When an I frame is received with a correct FCS, but its send sequence number, N(S), does not match the current receiver's receive state variable, the frame should be discarded. A REJ frame shall be sent with a receive sequence number equal to one higher (modulo 8) than the last correctly received I frame if an uncleared N(S) sequence error condition has not been previously established. The received state variable and poll bit of the discarded frame should be checked and acted upon, if necessary, before discarding the frame.

2.4.4.4 Reception of Incorrect Frames

When a DXE receives a frame with an incorrect FCS, an invalid frame, or a frame with an improper address, that frame shall be discarded.

2.4.4.5 Receiving Acknowledgement

Whenever an I or S frame is correctly received, even in a busy condition, the N(R) of the received frame should be checked to see if it includes an acknowledgement of outstanding sent I frames. The T1 timer should be cancelled if the received frame actually acknowledges previously unacknowledged frames. If the T1 timer is cancelled and there are still some frames that have been sent that are not acknowledged, T1 should be started again. If the T1 timer runs out before an acknowledgement is received, the device should proceed to the retransmission procedure in 2.4.4.9.

2.4.4.6 Receiving Reject

Upon receiving a REJ frame, the transmitting DXE will set its send state variable to the same value as the REJ frame's received sequence number in the control field. The DXE will then retransmit any I frame(s) outstanding at the next available opportunity conforming to the following:

Stellenbosch University http://scholar.sun.ac.za

1. If the DXE is not transmitting at the time, and the channel is open, the device may commence to retransmit the I frame(s) immediately.
2. If the DXE is operating on a full-duplex channel transmitting a UI or S frame when it receives a REJ frame, it may finish sending the UI or S frame and then retransmit the I frame(s).
3. If the DXE is operating in a full-duplex channel transmitting another I frame when it receives a REJ frame, it may abort the I frame it was sending and start retransmission of the requested I frames immediately.
4. The DXE may send just the one I frame outstanding, or it may send more than the one indicated if more I frames followed the first one not acknowledged, provided the total to be sent does not exceed the flow-control window (7 frames). If the DXE receives a REJ frame with the poll bit set, it should respond with either a RR or RNR frame with the final bit set before retransmitting the outstanding I frame(s).

2.4.4.7 Receiving a RNR Frame

Whenever a DXE receives a RNR frame, it shall stop transmission of I frames until the busy condition has been cleared. If timer T1 runs out after the RNR was received, the waiting acknowledgement procedure listed in 2.4.4.9, below, should be performed. The poll bit may be used in conjunction with S frames to test for a change in the condition of the busied-out DXE.

2.4.4.8 Sending a Busy Indication

Whenever a DXE enters a busy condition, it will indicate this by sending a RNR response at the next opportunity. While the DXE is in the busy condition, it may receive and process S frames, and if a received S frame has the P bit set to one, the DXE should send a RNR frame with the F bit set to one at the next possible opportunity. To clear the busy condition, the DXE should send either a RR or REJ frame with the received sequence number equal to the current receive state variable, depending on whether the last received I frame was properly received or not.

2.4.4.9 Waiting Acknowledgement

If timer T1 runs out waiting the acknowledgement from the other DXE for an I frame transmitted, the DXE will restart timer T1 and transmit an appropriate supervisory command frame (RR or RNR) with the P bit set. If the DXE receives correctly a supervisory response frame with the F bit set and with an N(R) within the range from the last N(R) received to the last N(S) sent plus one, the DXE will restart timer T1 and will set its send state variable V(S) to the received N(R). It may then resume with I frame transmission or retransmission, as appropriate. If, on the other hand, the DXE receives correctly a supervisory response frame with the F bit not set, or an I frame or supervisory command frame, and with an N(R) within the range from the last N(R) received to the last N(S) sent plus one, the DXE will not restart timer T1, but will use the received N(R) as an indication of acknowledgement of transmitted I frames up to and including I frame

Stellenbosch University http://scholar.sun.ac.za

numbered N(R)-1. If timer T1 runs out before a supervisory response frame with the F bit set is received, the DXE will retransmit an appropriate supervisory command frame (RR or RNR) with the P bit set. After N2 attempts to get a supervisory response frame with the F bit set from the other DXE, the DXE will initiate a link resetting procedure as described in 2.4.6, below.

2.4.5 Frame Rejection Conditions

A DXE shall initiate the frame-reset procedure when a frame is received with the correct FCS and address field during the information-transfer state with one or more of the conditions in 2.3.4.3.3, above. Under these conditions, the DXE will ask the other DXE to reset the link by transmitting a FRMR response as outlined in 2.4.6.3, below. After sending the FRMR frame, the sending DXE will enter the frame reject condition. This condition is cleared when the DXE that sent the FRMR frame receives a SABM or DISC command, or a DM response frame. Any other command received while the DXE is in the frame reject state will cause another FRMR to be sent out
with the same information field as originally sent. In the frame rejection condition, additional I frames will not be transmitted, and received I frames and S frames will
be discarded by the DXE. The DXE that sent the FRMR frame shall start the T1 timer
when the FRMR is sent. If no SABM or DISC frame is received before the timer runs out, the FRMR frame shall be retransmitted, and the T1 timer restarted as described in the waiting acknowledgement section (2.4.4.9) above. If the FRMR is sent N2 times without success, the link shall be reset.

2.4.6 Resetting Procedure

2.4.6.1

The resetting procedure is used to initialize both directions of data flow after a nonrecoverable error has occurred. This resetting procedure is used in the information-transfer state of an AX.25 link only.

2.4.6.2

A DXE shall initiate a reset procedure whenever it receives an unexpected UA response frame or an unsolicited response frame with the F bit set to one. A DXE may also initiate the reset procedure upon receipt of a FRMR frame. Alternatively, the DXE may respond to a FRMR by terminating the connection with a DISC frame.

2.4.6.3

A DXE shall reset the link by sending a SABM frame and starting timer T1. Upon receiving a SABM frame from the DXE previously connected to, the receiver of a SABM frame should send a UA frame back at the earliest opportunity, set its send and receive state variables, V(S) and V(R), to zero and stop T1 unless it has sent a SABM or DISC

Stellenbosch University http://scholar.sun.ac.za

itself. If the UA is correctly received by the initial DXE, it resets its send and receive state variables, V(S) and V(R), and stops timer T1. Any busy condition that previously existed will also be cleared. If a DM response is received, the DXE will enter the disconnected state and stop timer T1. If timer T1 runs out before a UA or DM response frame is received, the SABM will be retransmitted and timer T1 restarted. If timer T1 runs out N2 times, the DXE will enter the disconnected state, and any previously existing link conditions will be cleared. Other commands or responses received by the DXE before completion of the reset procedure will be discarded.

2.4.6.4

One DXE may request that the other DXE reset the link by sending a DM response frame. After the DM frame is sent, the sending DXE will then enter the disconnected state.

2.4.7 List of System Defined Parameters

2.4.7.1 Timers

To maintain the integrity of the AX.25 level 2 connection, use of these timers is recommended.

2.4.7.1.1 Acknowledgement Timer T1

The first timer, T1, is used to make sure a DXE doesn't wait forever for a response to a frame it sends. This timer cannot be expressed in absolute time, since the time required to send frames varies greatly with the signaling rate used at level 1. T1 should take at least twice the amount of time it would take to send maximum length frame to the other DXE, and get the proper response frame back from the other DXE. This would allow time for the other DXE to do some processing before responding. If level 2 repeaters are to be used, the value of T1 should be adjusted according to the number of repeaters the frame is being transferred through.

2.4.7.1.2 Response Delay Timer T2

The second timer, T2, may be implemented by the DXE to specify a maximum amount of delay to be introduced between the time an I frame is received, and the time the resulting response frame is sent. This delay may be introduced to allow a receiving DXE to wait a short period of time to determine if there is more than one frame being sent to it. If more frames are received, the DXE can acknowledge them at once (up to seven), rather than acknowledge each individual frame. The use of timer T2 is not mandatory, but is recommended to improve channel efficiency. Note that, on full-duplex channels, acknowledgements should not be delayed beyond k/2 frames to achieve maximum throughput. The k parameter is defined in 2.4.7.4, below.

Stellenbosch University http://scholar.sun.ac.za

2.4.7.1.3 Inactive Link Timer T3

The third timer, T3, is used whenever T1 isn't running to maintain link integrity. It is recommended that whenever there are no outstanding unacknowledged I frames or P bit frames (during the information-transfer state), a RR or RNR frame with the P bit set to one be sent every T3 time units to query the status of the other DXE. The period of T3 is locally defined, and depends greatly on level 1 operation. T3 should be greater than T1, and may be very large on channels of high integrity.

2.4.7.2 Maximum Number of Retries (N2)

The maximum number of retries is used in conjunction with the T1 timer.

2.4.7.3 Maximum Number of Octets in an I Field (N1)

The maximum number of octets allowed in the I field will be 256. There shall also be an integral number of octets.

2.4.7.4 Maximum Number of I Frames Outstanding (k)

The maximum number of outstanding I frames at a time is seven.

Stellenbosch University http://scholar.sun.ac.za

Appendix B

Implementation State Tables

The following state tables represent the implemented states & events and were drawn from the restricted description of the AX.25 protocol as discussed in Chapter 3. The tables are divided into three sections namely, *Command Frames Received, Response Frames Received* and *Miscellaneous Inputs.*

STATE	I	RR	REJ	SABM	DISC
S1 DISCONNECTED	NOP	NOP	NOP	UA,S5*	DM
S2 LINK SETUP	NOP	NOP	NOP	UA,S5*	DM,S1
S3 FRAME REJECT	NOP	NOP	NOP	UA,S5*	UA,S1
S4 DISCONNECT RQST	NOP	NOP	NOP	DM,S1	UA,S1
S5 INFORMATION TX	I	RR	I	UA	UA,S1
S6 REJ-FRAME SENT	I,S5	RR	I	UA,S5	UA,S1
S7 WAITING ACK	I	S5	I	UA,S5	UA,S1

NB. NOP : No Operation

* DM,S1 if unable to establish link

Table B1— State Table for Command Frames Received

STATE	RR	REJ	UA	DM	FRMR
S1 DISCONNECTED	NOP	NOP	NOP	NOP	NOP
S2 LINK SETUP	NOP	NOP	S5	S1	NOP
S3 FRAME REJECT	NOP	NOP	NOP	NOP	SABM,S2
S4 DISCONNECT RQST	NOP	NOP	S1	S1	NOP
S5 INFORMATION TX	I	I	NOP	SABM,S2	SABM,S2
S6 REJ-FRAME SENT	I	I	SABM,S2	SABM,S2	SABM,S2
S7 WAITING ACK	S5	I,S5	SABM,S2	SABM,S2	SABM,S2

NB. NOP : No Operation

Table B2— State Table of Response Frames Received

Stellenbosch University http://scholar.sun.ac.za

STATE	LOCAL START	LOCAL STOP	TIMER 1 EXPIRES
S1 DISCONNECTED	SABM,S2	NOP	NOP
S2 LINK SETUP	SABM,S2	DISC,S4	SABM
S3 FRAME REJECT	SABM,S2	DISC,S4	FRMR
S4 DISCONNECT RQST	SABM,S2	NOP	DISC
S5 INFORMATION TX	SABM,S2	DISC,S4	RR,S7
S6 REJ-FRAME SENT	SABM,S2	DISC,S4	RR,S7
S7 WAITING ACK	SABM,S2	DISC,S4	RR

NB. NOP : No Operation

Table B3— State Table Miscellaneous Inputs

STATE	N2 EXCEEDED	INVALID N(S) RECEIVED	INVALID N(R) RECEIVED
S1 DISCONNECTED	NOP	NOP	NOP
S2 LINK SETUP	S1	NOP	NOP
S3 FRAME REJECT	SABM,S2	NOP	NOP
S4 DISCONNECT RQST	S1	NOP	NOP
S5 INFORMATION TX	NOP	REJ,S6	FRMR,S3
S6 REJ-FRAME SENT	NOP	NOP	FRMR,S3
S7 WAITING ACK	SABM,S2	NOP	FRMR,S3

NB. NOP : No Operation

Table B3— State Table Miscellaneous Inputs (continued)

Stellenbosch University http://scholar.sun.ac.za

Appendix C

Protocol Flow Diagram

The following diagram represents the complete protocol flow diagram of the software implementation. There are some few points to be noted about the diagram:

- **T1** is the Acknowledgement Timer and is used to make sure that a station does not wait forever for a response to a frame it sends.
- **N2** is the maximum number of retries and is used in conjunction with T1.
- **DISC** is a disconnection command frame.

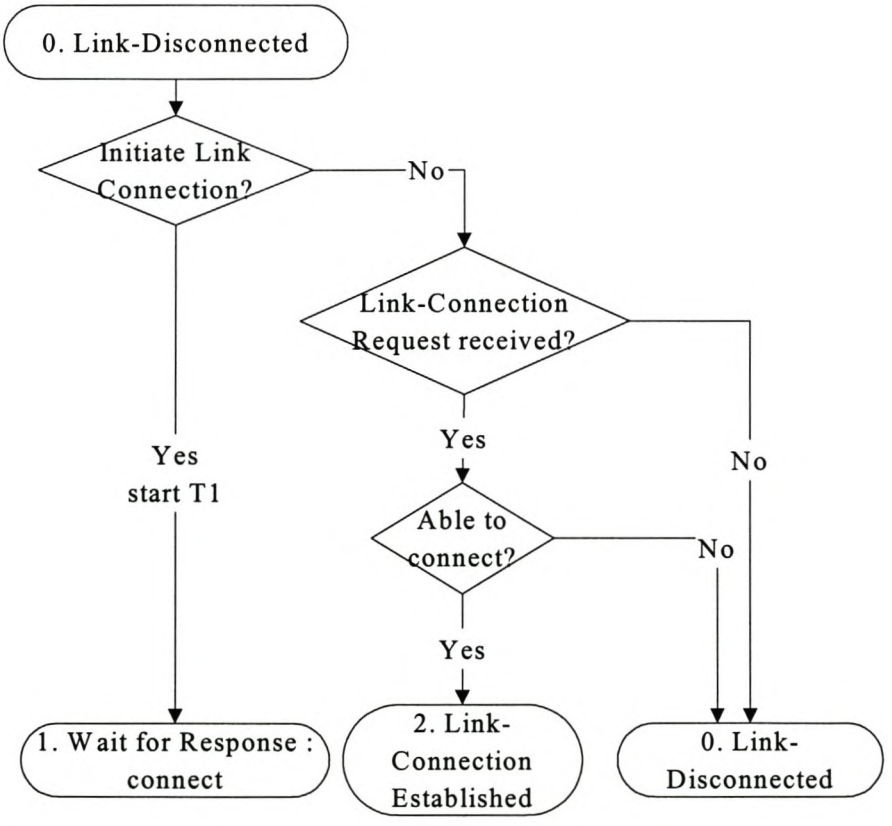


Figure C1: Complete Implementation Flow Diagram

Stellenbosch University http://scholar.sun.ac.za

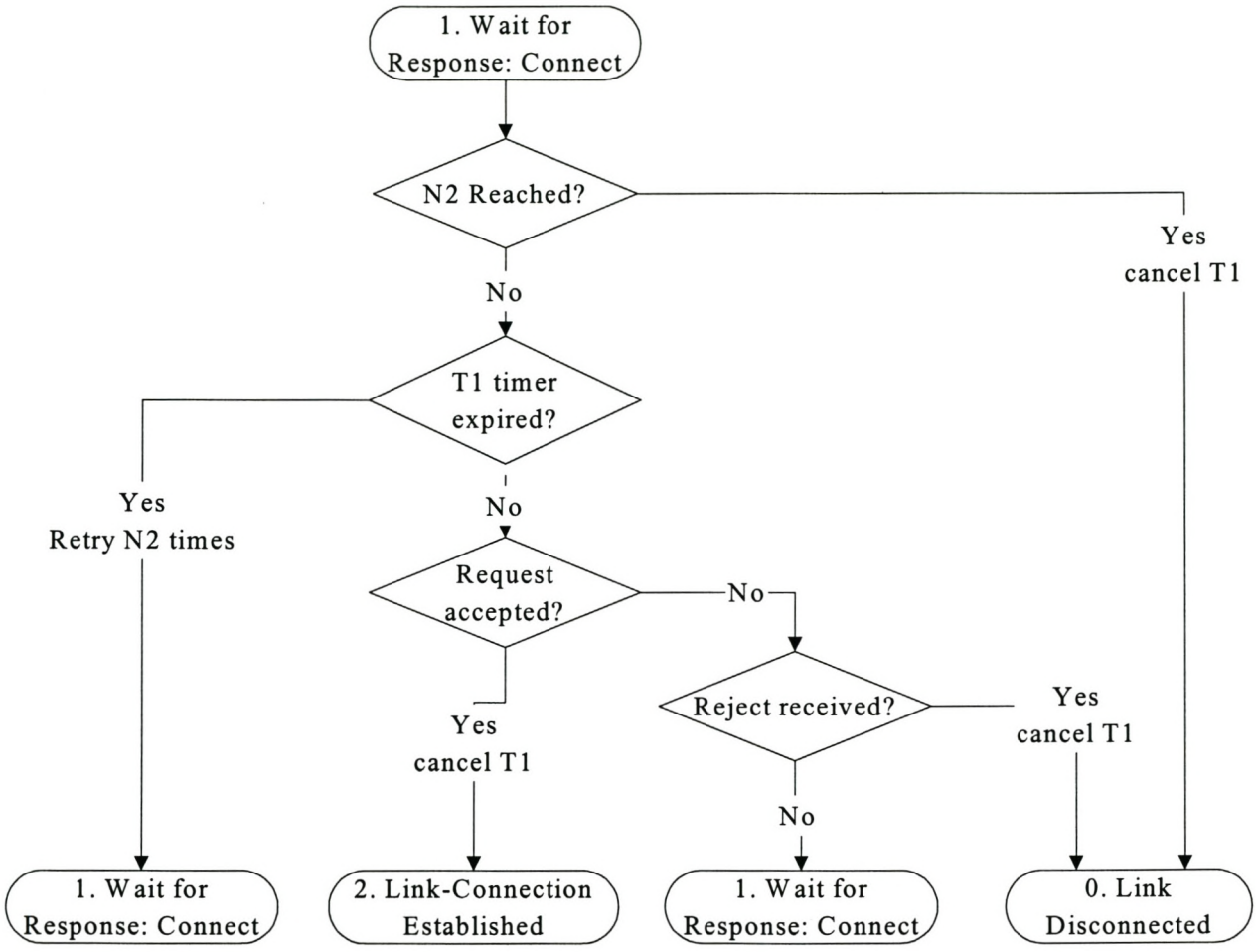


Figure C1: Complete Implementation Flow Diagram (continued)

Stellenbosch University http://scholar.sun.ac.za

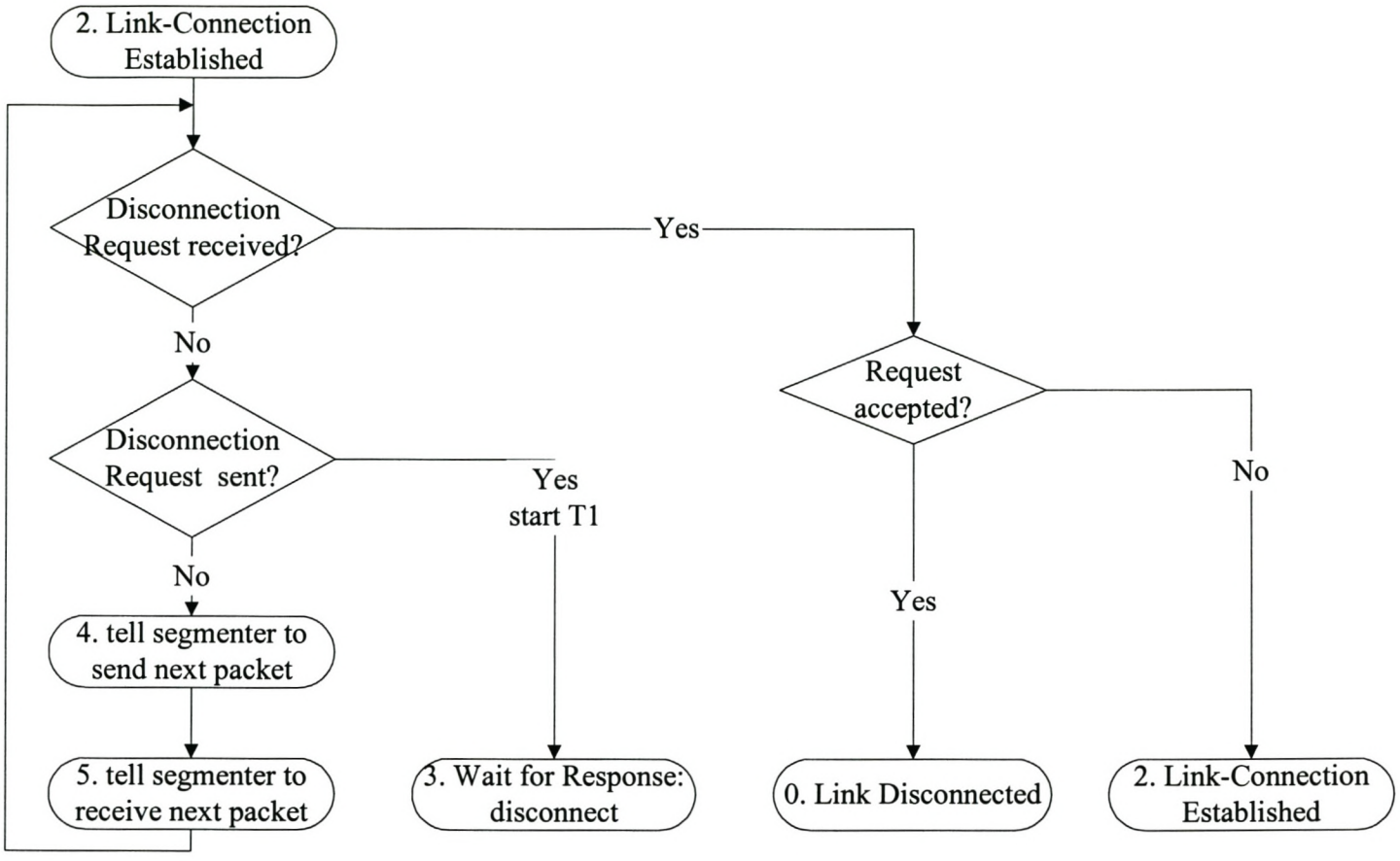


Figure C1: Complete Implementation Flow Diagram (continued)

Stellenbosch University http://scholar.sun.ac.za

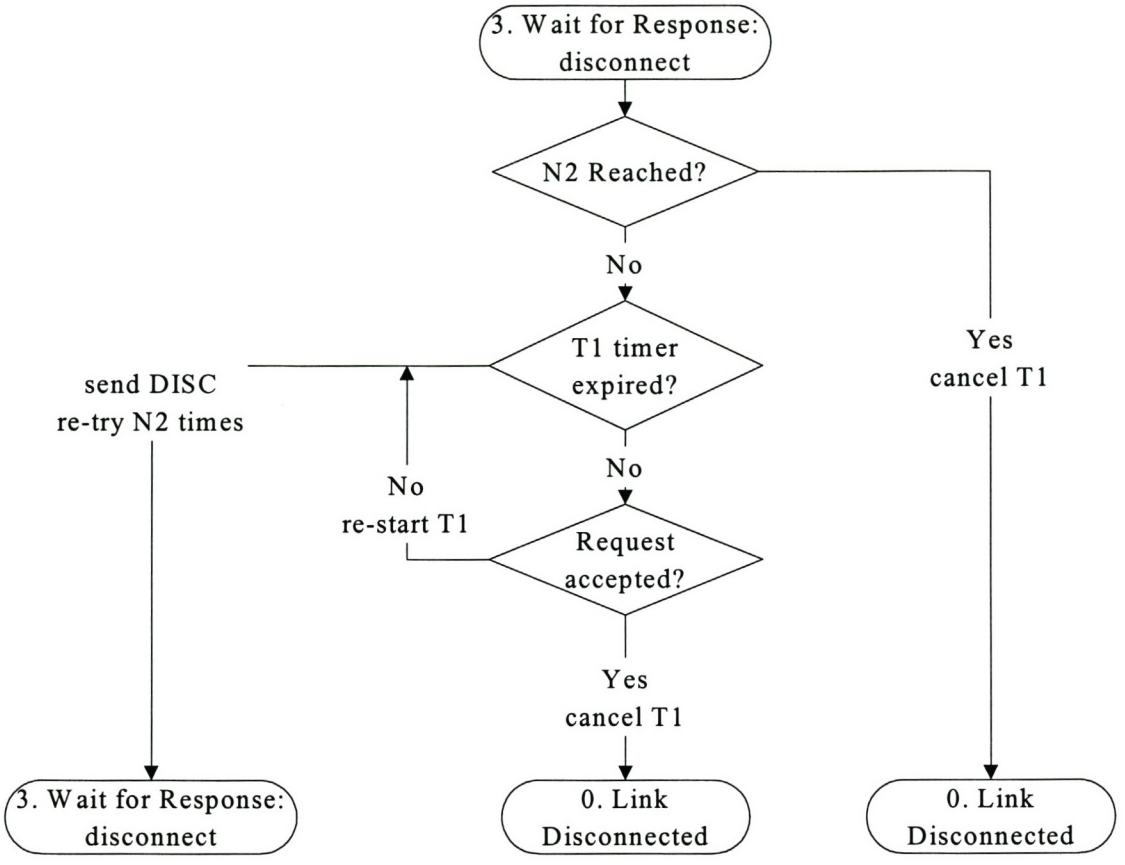


Figure C1: Complete Implementation Flow Diagram (continued)