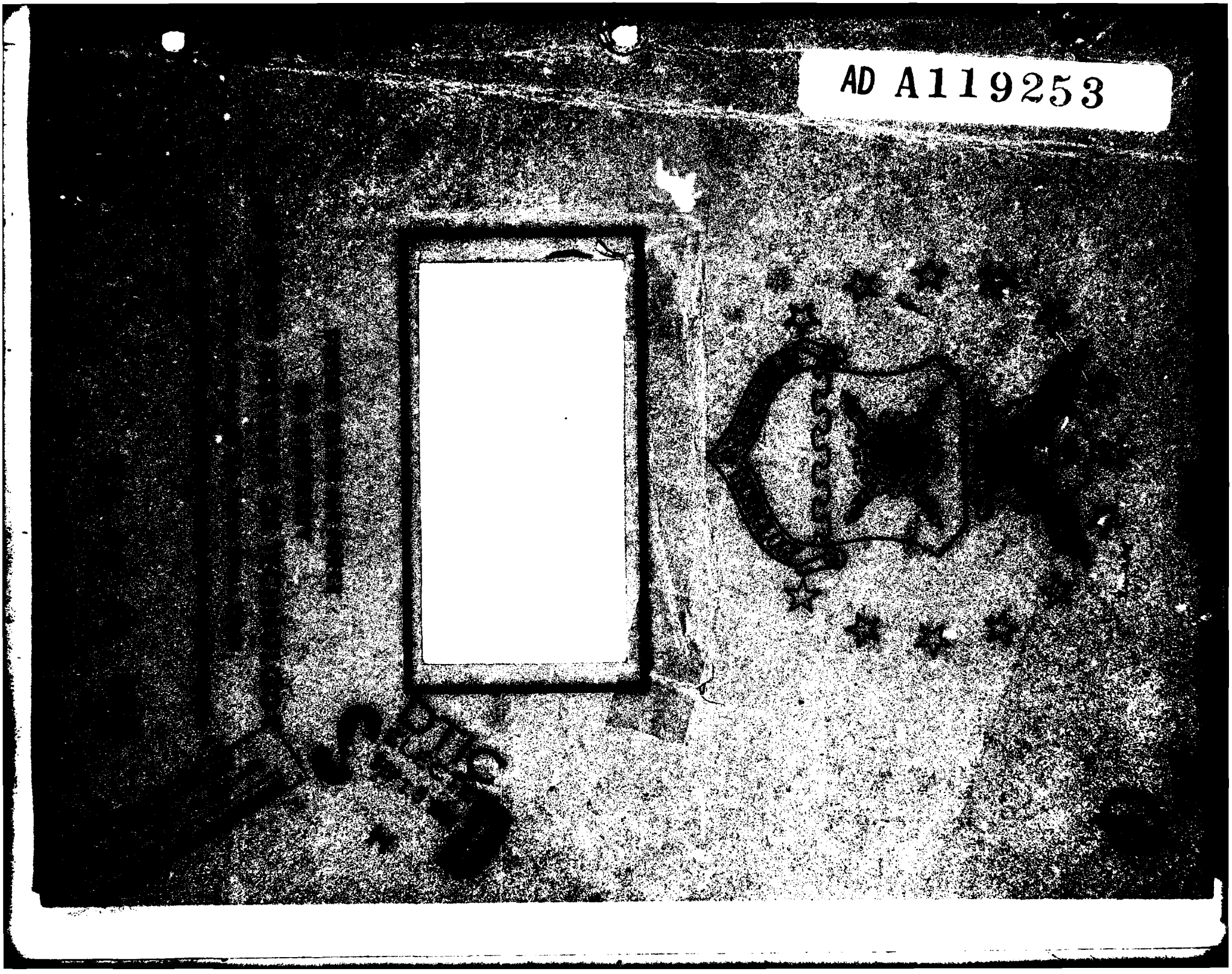
AD-A119 253 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONI--ETC(U)
DEC 81 R H STOKES
UNCLASSIFIED AFIT/GCS/EE/81D-16 NL



AFIT/GCS/EE/81D-16

DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONIC WARFARE DIVISION ENGINEERING BRANCH LABORATORY

THESIS

AFIT/GCS/EE/81D-16 Robert H. Stokes
CIV USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/81D-16

DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONIC WARFARE DIVISION ENGINEERING BRANCH LABORATORY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

by

Robert H. Stokes, BSEE, MSA
CIV USAF
Graduate Electrical Engineering

December 1981

Preface

This work presents a design of a local computer network for the Electronic Warfare Division Engineering Branch Laboratory which I hope will provide an engineering tool that will significantly reduce the ill effects caused by the reduced engineering manning levels currently existing within the branch. The design is based upon techniques developed by Tom DeMarco, and a network protocol model developed by the International Standards Organization. I gratefully acknowledge the part the above individual's and organization's techniques played in making my task simpler.

I would like to express my appreciation to Mr. Fred Massey and Mr. Steven Jones for their help in obtaining extensive documentation on the Digital Equipment Corporation's network, DECnet, Phase II and Phase III configurations at no cost to the government nor myself. I wish to thank Dr. Gary Lamont and Major Walter Seward whose leadership as my research advisors afforted me valuable guidance and encouragement. Also, I thank my reader, Captain James Moore for his constructive comments which helped to improve the clarity of this thesis. In addition, I am indebted to Mr. Joe Black and all the system lead engineers whom I interviewed to determine the requirements of the Engineering Branch Laboratory for a local computer network.

Finally, I wish to express my loving appreciation to my wife and family whose love and understanding endured through this graduate program.

Robert H. Stokes

Contents

Contents (cont.)

Contents (cont.)

List of Figures

List of Tables

List of Acronyms and Abbreviations

ACK	Acknowledge
ADD	Address
AFLC	Air Force Logistic Command
ARC	Area Reprogrammable Capability
ARPANET	Advanced Research Projects Agency Network
ATS	Advanced Threat Simulator
BPS	Bytes per Second
CCITT	International Telephone and Telegraph Consultative Committee
DEC	Digital Equipment Corporation
DECNET	Digital Equipment Corporation Network
DFD	Data Flow Diagram
DNA	Digital Network Architecture
ECF	Execute Command File
ECSAS	Electronic Countermeasures Signal Analysis System
ELINT	Electronic Intelligence Gathering
EMI	Electromagnetic Interference
EOF	End of File
EOS	End of Stream
ER	Erase
EW	Electronic Warfare
EWNET	Electronic Warfare Network
EWOLS	Electronic Warfare Open Loop Simulator
HOL	Higher Order Language
ISO	International Standards Organization
ISS	Integrated Support Station

K	kilo
M	mega
MTBF	Mean-Time-Between-Failure
MTR	Mean-Time-To-Restore
MTTR	Mean-Time-To-Repair
NAK	Negative Acknowledge
OFP	Operational Flight Program
OSI	Open Systems Interconnection
PACKETNETS	Public Packet Switching Networks
RF	Radio Frequency
RWR	Radar Warning Receiver
SCF	Submit to Command File
SISS	Standard Integrated Support Station
SISS SPEC	Standardized Integration Support Station Sys. Spec.
SISS STUDY	Standardized Integration Support Station Sys. Study
SNA	IBM's Systems Network Architecture
STG	Standard Threat Generator
TEWS	Tactical Electronic Wartare System
UNIVAC-1108	Electronic Warfare Installation Host Computer
WR-ALC	Warner Robins Air Logistic Center

Abstract

A local computer for the Electronic Warfare Division Engineering Branch Laboratory was designed around a commerically available and supportable network configuration. The requirements for this network were specified by interviewing the engineers associated with the Engineering Branch Laboratory and then translating the functional requirements into a detailed set of hardware and software system requirements. Structured Analysis was used to produce a structured specification for application, transport, network, and data link protocol level requirements. Digital Equipment Corporation's 'DECnet' Phase II and Phase III network configurations were combined together to form this unique Electronic Warfare Network (EWNET). The node network uses a loop topology with a star of up to seven hosts connected to each node. The nodes are implemented using Digital Equipment Corporation's PDP-11/70 computers. Initially, the network will include fourteen Integrated Support Station computers which are either Digital VAX-11/780s or Digital PDP-11/34s. These computers will be connected to the nodes using duplex fiber optic links supporting transmission rates up to 1 Mbs. The Phase II DECnet protocol was selected to provide the file transfer and data transport protocols in conjunction with a basic routing algorithm at each host, while the Phase III DECnet protocol was selected to provide these functions at a higher level in the nodes. The selection of the above topology in conjunction with the described

protocol structure keeps any one host from degrading the network if the host should fail. All Integrated Support Station common off-line functions, common databases, and commonly used support software tools are hosted on the node computer for easy, universal access, allowing for a degree of standardization.

I. Introduction

The purpose of this thesis investigation was to design a local computer network for the Warner Robins Air Logistics Center (WR-ALC) Electronic Warfare Division Engineering Branch. This network, named the Electronic Warfare Network (EWNET), was first proposed in 1980, when the number of Integrated Support Stations in the Engineering Branch had grown to nine. At this time, there were insufficient peripheral devices plus engineering expertise to completely support existing and future electronic warfare (EW) efforts. Local networks were becoming invaluable aids to organizations and corporations similar in structure to the EW Division. This was due to a desire to increase information processing power without additional computers. Also, corporations were finding that the shortage of electronic and software engineers was causing project non-support to result since multi-system expertise amoung existing engineers was becoming next to impossible to obtain. Interest in both of the above areas plus the potential cost savings, provided the impetus for the development of a local computer network for the WR-ALC EW Division.

Historical Perspective

For the past decade, the field of Data Communications has been rapidly changing, with important innovations emerging all the time. One

recent development of wide ranging significance is the introduction of several new techniques for short distance high speed local computer networks. By definition, a local network is a data communication system designed to interconnect computers and terminals over a restricted geographical area, typically less than 2-kilometers in diameter (Ref. 4:18). A number of implications follow from this definition.

At a technical level, the problems involved in designing local networks are not very different from those relating to long distance networks such as the Advanced Research Projects Agency Network (ARPANET). However, the parameters are different. Since communication is required only over a local region, the following observations can be made.

-- A more expensive communication medium, in terms of cost per meter, can be used in a local network because the total cost of the medium is likely to be small compared to installation and other hardware and software costs.

-- Since a more expensive communication medium can be used, a more powerful medium is possible, especially in terms of speed and error performance. Thus the communication network and communication medium represent less of a bottleneck to the system as compared to geographical (global) networks.

-- Since the communication network and transmission mediums are no longer bottlenecks, transmission rates are higher. Delivery rates and maximum delivery delays become shorter. The increased

bandwidth and shorter transmission distance of the communication medium causes error rates to be reduced. Also, the short transmission distance reduces the cost associated with interconnecting terminal and computer equipment.

-- Since wide bandwidth mediums are cost effective for use with local networks, greater use of broadcast or multi-address communications is possible. The wide bandwidth medium alleviates the problem of channel contention where two or more transmitting terminals overlap on the same channel (clash).

-- Local interface hardware and communication protocols can be considerably simplified because there is no need to optimize available communications bandwidth, since the inherent traffic handling capacity of the network is so much greater than in a global network.

-- The cost of interconnecting a device to a local network can be reduced by the procedures of simplification (no longer need central switching or control systems) and standardization.

Only recently have local networks been used extensively, therefore no standardization exists. One of the most attractive possibilities for standardization is the problem of a standard interface and associated protocol. For example, the X.25 Protocol, which defines the interface of a computer to a packet-switching network, has just recently become a standard (Ref. 8:108). Additionally, one of the most often ignored

problems when considering the design of a local network is the number of levels of function-oriented protocols necessary to accomplish useful work on the network. In order to be useful the local network must support some form of data transport protocol, file transfer protocol, and virtual terminal protocol. These protocols are necessary so that a computer which was designed with one data and file format, and with a particular terminal type in mind, can transfer data and files to other computers in other formats, and accept information from foreign terminal types (Ref. 4:26).

Finally, no standard local computer network exists. In the past, most local computer networks available commercially were developed by a firm to support their own systems. Foreign device interfacing was accomplished through the development of an emulator to make the foreign device appear at the interface as one of the manufacturer's own devices. IBM's System Network Architecture (SNA) and the Digital Equipment Corporation's Network (DECNET) are both examples of this type of network (Ref. 1:3).

Recently developed by Xerox Corporation, the ETHERNET network concept, utilizing contention-allocation mechanisms (packet collision detection) and data packets on a logical channel, can interface a hetergeneous set of computers together (Ref. 5:78).

Thus local computer networks have evolved from long-distance networks to meet the needs of a local organization requiring internal control of their own systems. The local network offers advantages not

accessable to organizations in the past. Even though problems still exists, several existing systems such as DECNET, SNA, and ETHERNET have come a long way in providing a standard design. So it is from this viewpoint that the development of EWNET took place. The following sections give additional definitions and background necessary to understand the EWNET design decisions.

Background

This investigation follows two studies performed by the Georgia Institute of Technology Engineering Experiment Station Systems Engineering Laboratory (Ref. 2,3). The Standardized Integration Support Station System Study (SISS Study) was the final standardization report which identified the need for the EWNET (Ref. 2:8-13). The Standardized Integeration Support Station System Specification (SISS Spec) was the final system specification that resulted in several general theoretical concepts upon which the network design was based (Ref. 3:41-43).

Both documents cover the EWNET in limited detail and essentially state a reason for the network followed by a possible configuration. The configuration presented in the SISS System Specification is based on the author's personal preference and not on an actual study of the Engineering Branch requirements. Therefore, additional work was required before a design for EWNET could be developed and implemented in the Engineering Branch Laboratory.

Integrated Support Station

To correctly design a local computer network, the functions to be supported by the network must be understood in context with the existing support equipment.

Integrated Support Stations (ISS) are systems that support the efforts required to provide software reprogramming and hardware update prototyping for electronic warfare (EW) systems. An ISS is not used to repair or test the numerous fielded EW systems of a given type.

An ISS is divided into two major parts. For convenience, these two parts will be referred to as the ISS Development System and the ISS Control System. One system controls the testing of an operational EW system modified for laboratory use (Hot Mock-up). The other major system supports the development of the software for the EW system. A functional block diagram is shown in Figure 1 on page 8.

The ISS Development System supports reprogramming development for the EW system Operational Flight Program (OFP) software. The facilities provided for development include facilities to prepare OFP software for the EW system and computers in the ISS, to support distribution of OFP software to the field, to prepare test procedures for the EW system, and to analyze results of EW system tests.

The ISS Control System provides supervisory control of the ISS, automatic execution of EW system test including stimulus of the EW system Hot Mock-up and response measurement system, an automatic test

and calibration of ISS stimulus and measurement equipment, and communication facilities in the ISS and to the network of all ISSes and other computers.

Each ISS interfaces with the Advanced Threat Simulator (ATS) and the Electronic Warfare Open Loop Simulator (EWOLS) and the Electronic Countermeasures Signal Analysis System (ECSAS) for the purpose of performing large scale tests of the supported EW system. The EWOLS or ATS and ECSAS combination is a dense radio frequency (RF) signal enviroment and analyzer, respectively (Ref. 3:4).

Each ISS has the capability to operate independent of the ISS network and any common large scale computer. Therefore, an ISS is a support station that is built around an advanced state-of-the-art minicomputer with special internal interfaces to the Hot Mock-up to control and monitor OFP software and flight hardware operation.

Objective of This Investigation

The objective of this investigation was to specity the design of the EWNET in "sufficient" detail to obtain a network design that met the Engineering Branch baseline need of a local resource sharing network that is highly reliable, through built-in redundancy, and contains a minimal amount of one-of-a-kind technology. To arrive at this baseline design, the following steps were followed:

1. Survey of user needs and requirements.

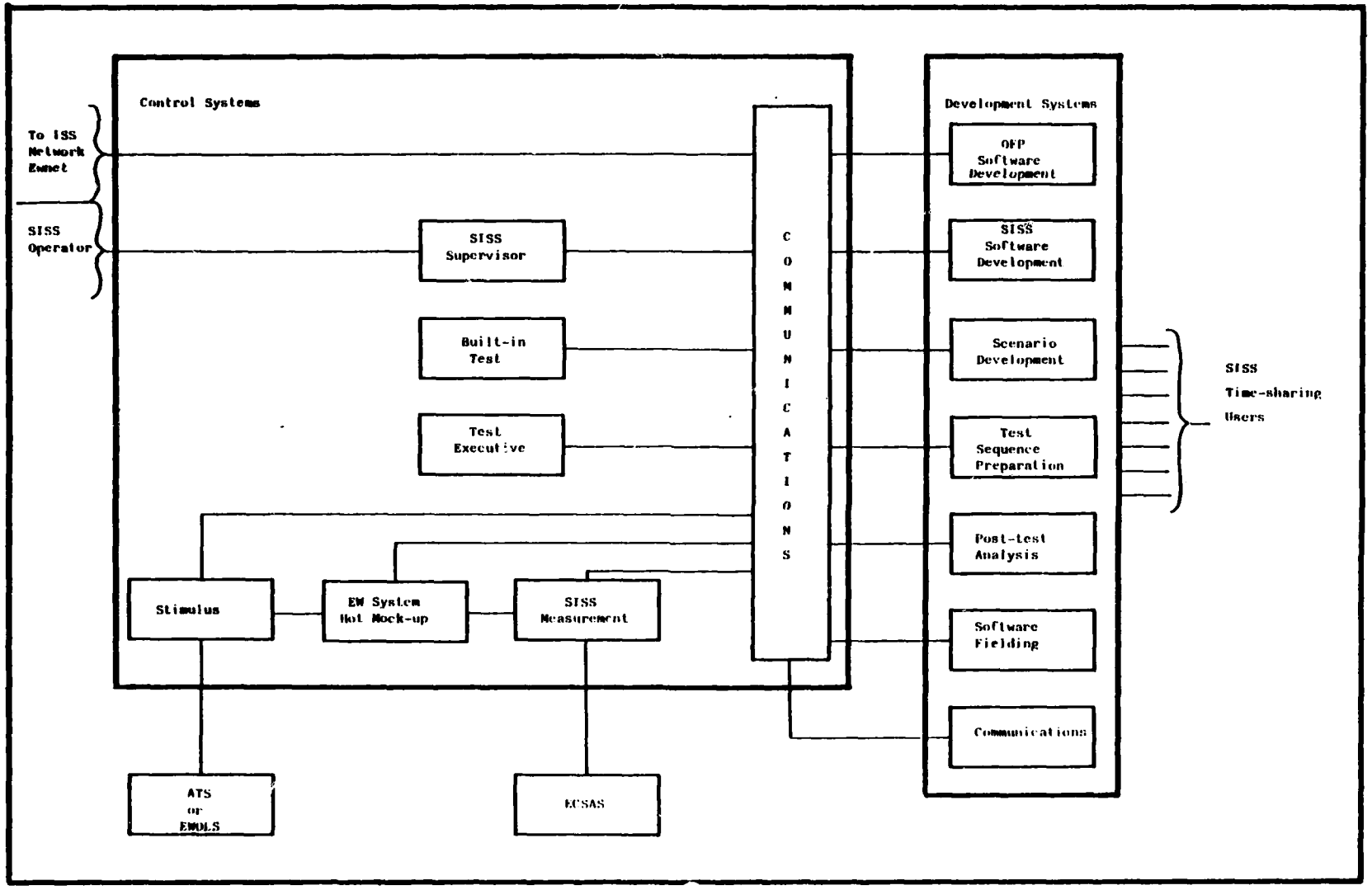


Figure 1 SISS Functional Block Diagram

2. Determination of user functional requirements.

3. Translation of functional requirements into a system specification through the use of Structured Analysis techniques (Ref. 9:55-165).

4. Development of the hardware design

5. Specification of the necessary protocol within a commercially available system software design and

6. Evaluation of the Electronic Engineering Branch Laboratory Host processor (UNIVAC-1108) off-line functions to determine if the functions can reside within the EWNET configuration.

Approach

The following approach was used in the design of EWNET. First, a top-down development of the design was chosen. This allowed the design to first address those levels closest to the user, thereby insuring that the design would be consistent with the user's requirements. Then the lower levels of the design were developed to support the requirements of the next higher levels.

Due to the size of the development effort, structured analysis and design techniques were used in the approach to provide clear written specifications and diagrams. Structured analysis and design techniques were chosen over other techniques since they provide unambiguous

information about a large design to those implementing it.

The first phase of the investigation consisted of researching the literature and gaining a working knowledge of computer networks, protocols, hardware interfaces, and the capabilities of the Engineering Branch Laboratory ISSes.

Once a sufficient background had been obtained, a user interview was designed and the ISS system lead engineers were interviewed. From these interviews, a set of projected uses and functional requirements were compiled and used to determine the system requirements.

The system requirements consist of both hardware requirements for the topology, hosts, nodes, and transmission mediums, as well as software requirements. Structured analysis was used to document the software requirements (Ref. 9). *Data flow diagrams* and a data dictionary were used to generate the structured specification.

The system requirements were then used to develop a hardware design and a software design through the study and refinement of an existing commerically available network configuration. The final stage of the investigation included an evaluation of the Engineering Branch UNIVAC-1108 functions (off-line support, backup, and databases) and the resulting decision to delete the Branch requirement to perform the UNIVAC-1108 functions on the UNIVAC-1108 computer. Instead, these off-line functions will, in the future, be performed by the EWNET node computers.

Overview of the Thesis

The structure of the thesis basically follows the approach that was taken in the investigation.

Chapter II contains an analysis of the EWNET functional requirements. Appendix A provides the supporting information including the compilation of the results of the user interviews, ISS systems to be supported by EWNET, a list of those interviewed, and their respective areas of expertise.

Chapter III translates the functional requirements into hardware and software requirements. Structured analysis is defined and data flow diagrams are used throughout the chapter to support the written description of the software requirements. Appendix B contains the floor layouts of the location of the EW systems to be supported by EWNET with tentative locations identified for the EWNET node computers. The Georgia Institute of Technology proposed topology diagrams are included in Appendix B to further clarify the floor layouts (Ref. 2:10). Appendix C contains the complete structured specification in support of the software requirements.

Chapter IV describes the design phases of EWNET. The design of the hardware is specified, including the topology to be employed, the EW systems to be included in the network, the nodes to be used in the network, and the transmission medium to be used. The software design is described in terms of a commercially available network design. First, the functional specification and requirements were used to select the

best commercially available network from a list of three existing designs. Then the selected design was tuned to represent the exact requirements of EWNET. Additionally, off-line host software was allocated to the node processors. Finally, Appendix D contains the actual software and hardware design in the format required by the selected vendor, so that the Appendix D can be used as the document necessary to procure the EWNET.

Chapter V summarizes this investigation and gives recommendations for follow-on research efforts.

II. EWNET Functional Requirements

Introduction

This chapter specifies the functional requirements of EWNET. First, the background of the requirements analysis is discussed, including the reason it is important to conduct a thorough investigation of the functional requirements. Then the content of the user survey that was used to help determine the EWNET requirements is described. Also included in this section is a description of how the survey was conducted including personal observations about the survey contents. The next section summarizes the projected uses of EWNET, followed by sections summarizing user-oriented, design-oriented, and additional functional requirements. Finally, EWNET constraints not addressed by the user survey are discussed.

Background

Possibly the most crucial step in designing a computer network is the specification of the network requirements. Yet, this phase of development was completely overlooked by both the SISS Study and the SISS System Specification (Ref. 2,3). While it is possible to specity many requirements that are generally desirable in a network, using this approach without additional thought will result in a incomplete and inaccurate requirements specitications. This is due to the way local

computer networks vary from organization to organization (Ref. 4:18). This is obvious from the many varied designs that presently exist in industry. Also, not all requirements for a network have the same weight. Due to time or money constraints, the network may not be able to support all requirements. Some requirements add no value to the network when considered for a particular organization's uses. Finally, every organization is unique and would have unique requirements for 'their' network. These unique requirements would be overlooked if we only considered what is generally desired in a network.

User Survey

A systematic approach was needed to specify the requirements for EWNET that would tailor it to the requirements of the Engineering Branch Laboratory. Thus, a four-part user survey was designed.

In the first section, the users were asked what applications could be served by EWNET from their utilization perspective. To aid the users being interviewed, twelve common local network applications were listed which the user could evaluate on a five-point scale from 'OF-NO-USE' to 'VERY GOOD'. These applications included peripheral sharing amoung the computers in the network, file accessing and transfers across the network, sharing software tools on the network, accessing AUTODIN I, accessing ARPANET, accessing the UNIVAC-1108 host installation computer, using digital threat information generated from a central location, monitoring Eglin AFB flight test in real time, doing distributed

processing, managing distributed databases, monitoring Eglin AFB flight test video, and providing fault tolerance for the EWNET. Finally, this section asked the users to identify the applications that they felt should be implemented first and to prioritize the applications in terms of their personal needs with any related comments.

The second section asked the user to estimate the requirements from their perspective for six basic network parameters. The user-oriented parameters were throughput, response time, the user interface, security, availability of the network, and ISS-to-ISS interaction. Each of the subsections addressing a parameter had a number of questions to aid the user in evaluating the requirement for that parameter. Each user was then asked to identify any other user-oriented parameters that might influence EWNET's design. Finally, the second section concluded by having the user rate each of the six parameters on a five-point scale from 'DOES NOT APPLY' to 'MUST HAVE'.

The third section asked users to make any other comments that they felt might help specify the requirements of EWNET.

The fourth section asked the general branch management to estimate the design-oriented functional requirements of EWNET since no single network management group existed within the Engineering Branch. Three basic network design-oriented parameters were listed. They were flexibility, performance monitoring, and the availability of a distributed processing language. As with the user-oriented parameters, each subsection contained a number of questions to help management to

evaluate the parameter. Management was also asked to identify other design-oriented parameters and conclude the section by rating each of the three parameters on a five-point scale from 'DOES NOT APPLY' to 'MUST HAVE'.

The user's of EWNET were addressed as being the ISS system lead engineers, section leaders, and the branch chief. The users were sent the 'EWNET INTERVIEW' survey two weeks prior to the actual interviews being conducted. This was done to give the users time to develop their own questions about the survey.

Each user was interviewed for approximately fifty minutes using the user survey forms sent to them two weeks prior. Personal interviews were selected to allow the user to completely understand the survey by asking direct questions to the interviewer. Also, the interviewer was able to help the user weigh his answers to the survey questions. The response to the user interviews was excellent and the information obtained was used to formulate the general requirements specifications that follow.

Projected Uses of EWNET

The initial uses of EWNET as presented in The Standardized Integration Support Station System Specification" are as follows (Ref. 3:8):

1. Threat data access = access by every ISS to the Standard

Threat Database hosted on the Electronic Warfare Branch Host Computer (UNIVAC-1108).

2. Hardware resource sharing = ability to use other ISS's hardware through the use of the network.

3. Software resource sharing = ability of each ISS to use a common depository of software tools.

4. Supplemental analysis = The ability of each ISS to perform off-line time-consuming supplemental analysis processing (flight-test data reduction) on the Electronic Warfare Branch host computer.

5. Back-up processing = in the event of a complete ISS computer failure, the ability to continue with EW support through the use of the network and the Electronic Warfare Branch host computer. Finally, the network will be used as one of the instruments to bring about complete standardization of the Engineering Branch Laboratory.

The user survey results were used to determine the projected uses of EWNET. The principal uses that were considered most beneficial were in the area of resource sharing. Peripheral sharing, file access and transfer, software tool sharing, and access to the UNIVAC-1108 were identified by all users as being of top-priority. Next in potential benefits was the capability of executing job processes that can run concurrently on different computers (distributed processing) and the

capability to access and maintain distributed databases. Access to ARPANET and to AUTODIN I through the use of EWNET was completely rejected because the security of the Engineering Branch allowed no external (outside immediate building area) gateways to the ISSes. Access to Eglin AFB flight test data (digital and video) was looked upon as a very nice capability but not required for the immediate future due to the cost incurred in obtaining a dedicated satellite plus encryption and decryption devices, receivers, and transmitters. Access to a remote digital threat generator was rejected because most ISSes already have a local capability and those that do not are in the process of developing this capability. Finally, the network is expected to support critical functions, therefore, the users perceived that a need would arise for the network to guarantee uninterrupted service to the ISSes through redundancy and fault-tolerance. The projected uses of EWNET are summarized in Table 1.

User-Oriented Functional Requirements

The user's functional requirements were also obtained from the survey. The throughput indicated by the users required the data rates in the network to be a minimum of 1 million bits per second per transmission link. Data rates between the Host computer and the other computers in the network should be a minimum of 56,000 bits per second per transmission link. Usage of the computers in the Engineering Branch Laboratory will average between 6 to 10 hours per day with peaks of up to 24 hours a day for all ISSes during official OFP change exercises.

These exercises are staggered

PROJECTED USE	VERY GOOD	GOOD	MEDIUM	LITTLE USE	NO USE
PERIPHERAL SHARING	5	4	2	1	0
FILE TRANSFERS	8	2	1	1	1
SOFTWARE TOOL SHARING	8	3	1	0	0
ACCESS TO UNIVAC-1108	5	4	1	1	2
ACCESS TO ARPANET	0	0	0	4	8
ACCESS TO AUTODIN I	1	1	2	0	8
THREAT GENERATOR	4	5	1	3	1
FLIGHT TEST MONITOR	2	1	3	3	3
DISTRIBUTED PROCESSING	2	3	4	3	0
DISTRIBUTED DATABASES	2	3	4	4	0
VIDEO	1	2	3	2	4
FAULT TOLERANCE	4	7	2	0	0

Table 1 Projected Uses of EWNET (Composite of User Responses)

throughout each year and usually only two thirds of all ISSes are affected. The ISS computers being affected by heavy usage at the present time are listed below.

1. Harris 6024/4

2. VAX 11/780

3. PDP 11/34

4. Modcomp Classic

5. Data General Ellıspe S-230

These ISS computers are the presently used computers. Future standardization plans (by 1984) require that all ISS host computers will be one of the models from the Digital Equipment Corporation line of computers (Ref. 3:17-20).

The response time requirement was, of course, closely tied to the throughput requirement. The requirement for response time was addressed for three modes: interactive, file transfers, and echoing user inputs. In the interactive mode, two to three seconds for 'simple' commands was considered to be satisfactory. For file transfers, the response time was set to five minutes for a 32 Kbyte file. Also, each ISS will require the ability to obtain and verify a copy of the Standard Threat Database from an installation host computer through the network within 20 minutes. This was considered an acceptable tradeoff between user requirements and the file accessing capabilities of the computers in the Engineering Branch Laboratory. The echo response time was given as a maximum of one-half second. This was to insure that the echos of the user inputs did not interfere with their entering subsequent data at the terminal keyboard. All response time requirements seemed consistent with the results of psychological studies predicting satisfactory levels of performance for the typical user (Ref. 6:322-323). An additional

requirement on the response time addressed consistency in the response times.

Other user interface requirements of equal importance included error recovery, built-in-test, user 'help' command capabilities, and an in-house distribution capability so that messages from one ISS to another could be transmitted point-to-point. Also, a management distribution capability was recommended. In this case a message would be sent from someone at an ISS to perhaps a stand-alone terminal located at a central management point. All users stressed the importance of maintaining ISS independence from the network in case there was a network-wide failure the individual ISSes could continue to function. Finally, all users indicated that ISS-to-network accounting data would be a requirement, especially if it would provide them with the number of times their ISS was being externally accessed over any given day.

Security was addressed from four perspectives. All users indicated that 'Secret' data would be running on the network and on the ISSes. Therefore, the concensus was that the network would require a design to safeguard the processing of classified data. The second aspect of security addressed was the requirement to protect files on the network from unauthorized access or alteration. It is highly undesirable for all ISSes to have unlimited access to all classified files without some means of control and need-to-know establishment for the particular data. Thus, file access restrictions were found to be a security requirement for the network. A two element file password to protect this need-to-know was suggested. And as a matter of security accountability

it was suggested that the ISS/network interface accounting package should record time, destination, and password of all classified recipients (each access). This would also be of benefit to audits of ISS classified activity when problems arose. The third aspect of security was related to the electromagnetic interference (EMI) generated by the network mediums. All users indicated that fiber optic based systems should be used as the network medium, where possible, because fiber optic based systems are less susceptible to EMI than systems using traditional metallic connections. The fourth and final aspect of security was access control to the network from outside the Engineering Branch Laboratory. As discussed before, all users indicated that such gateways could not be allowed due to the restrictions placed on the ISS internal security.

The availability of EWNET was defined to be the percentage of time that the network provided the capabilities required by a particular user as compared to the time that it was suppose to provide those capabilities. This definition resulted in assessments ranging from 50 percent to 100 percent with 100 percent being the answer given by almost half of the users. The time periods that the network should be available were identified as either normal duty hours (0800-1700) or during times when emergency OFP changes would require duty to extend beyond normal hours. The 90 percent availability level was finally determined to be acceptable considering the network should be available 90 percent of a 24 hour day, leaving 10 percent of the 24 hour day for maintenance downtime. This 90 percent availability level represents a

reasonable target for the system design.

ISS-to-ISS interaction was considered to be of minimual importance. Most users felt that ISS-to-ISS interaction within a section (type of ISS) would be beneficial but that external section interaction would not be needed. This internal section ISS-to-ISS interaction directly corresponds to the suggestion that ISS-to-ISS interaction take place only around a node computer and/or with the installation host computer.

Other user-oriented requirements included a word processor capability plus central network documentation located on a installation host computer. A common network command language was also thought to be important to minimize confusion. A real-time network capability where one ISS took complete control of the network when emergency changes dictated it was suggested. Finally, for future consideration, a data link to EW Elint sources, overflow / surge load handling (using another ISS's terminals to operate your ISS), and self-configuration through periodic status checks of the network members were other suggestions made but were felt to be of little immediate importance. Table 2 summarizes the user-oriented functional requirements.

Design-Oriented Functional Requirements

The design-oriented functional requirements included flexibility, performance monitoring, and distributed processing language. The flexibility requirement was addressed by asking management to describe how they would see the network changing over the next five years. The

AREA	MUST HAVE	VERY APPLY	APPLY	MARGIN APPLY	NOT APPLY
THROUGHPUT	3	7	3	0	0
RESPONSE TIME	3	7	2	1	0
USER INTERFACE	8	4	2	0	0
SECURITY	12	0	0	0	0
AVAILABILITY	5	4	4	0	0
ISS-TO-ISS INTERACTION	1	2	1	4	4

Table 2 User-Oriented Functional Requirements (Composite of User Responses)

response given by management was that more nodes and ISSes would be added to the network. Management felt that it was very important for EWNET to be easily reconfigurable with respect to adding additional nodes and ISSes. Other changes mentioned included increased use of the network for management functions, and a transition of the network workload from predominantly file transfers to more interactive traffic. All users expressed the concern that the network should start small with limited capabilities and as new uses are identified as being valuable to the organization, the changes could be implemented into the network with minimum effort. Flexibility with respect to the topology, protocols, and transmission medium was considered to be important only to the extent of the changes necessary to these areas when additional nodes and ISSes were added to the network.

The need for some form of performance monitoring capability was expressed by management, but they felt that the limited knowledge on

their part would only confuse the matter. Generally, management felt that accounting data should be available on the network plus node statistics, a software monitor, and a way to detect network bottlenecks. The requirement for a performance monitoring node was rejected, management felt that to protect the network from slowdown that performance monitoring should be done only on a demand basis.

The need for a distributed processing language has been a topic of much discussion between the Air Force Logistic Command Headquarters and the Electronic Warfare Branch at Robins AFB. The Embedded Computer Standardization Program Office established 7 January 1981, as the single Air Force Office responsible for the acquisition of software tools for users of computers designed to MIL-STD-1750A (Instruction Set Architecture) is to provide support to Air Force Logistic Command (AFLC) to develop a Jovial J-73 compiler for the VAX 11/780 computer. This office and Jovial J-73 will be used to assist the Air Force in the transition from currently used higher order languages (HOL) to the tri-service HOL of the future - ADA (Ref. 7:3). Due to the above, management feels that all ISS host computers will be required to implement Jovial J-73, Fortran, and eventually ADA programming languages on the network host computers. ADA will become the distributed processing language at WR-ALC. Table 3 lists how management ranked each of the design-oriented functional requirements on a scale from 'DOES NOT APPLY' to 'MUST HAVE'.

AREA	MUST HAVE	VERY APPLY	APPLY	MARGIN APPLY	NOT APPLY
PERFORMANCE MONITORING	2	3	4	0	0
DIST. PROCESS LANGUAGE	2	1	6	0	0
FLEXIBILITY	4	3	2	0	0

Table 3 Design-Oriented Functional Requirements (Composite of User Responses)

Additional Functional Requirements

The Third section of The user interview was used by users and management alike to list additional requirements for the network. The ability to find a file in the network that had been sent from one host to another was felt to be an important requirement. The network being used for general software development and the network implementation being started with limited capabilities, planning for future growth, were two requirements that were well accepted. Requirements such as being able to initialize the network with an arbitary subset of nodes, the location and number of nodes, and the number of ISSes per node were areas where most users had no established opinions. As a final requirement, management felt that the network access should be protected through the use of a network password.

Constraints on EWNET

The network described in this thesis will be required to operate successfully in an enviroment characterized by significant levels and

amounts of electromagnetic interference (EMI). All network equipments, systems, subsystems, and modules shall be able to perform as specified within the EMI enviroment while not permitting any performance degradation or erroneous data and signals to be introduced. MIL-STD-461 should be used as a guideline.

Additionally, the communication subsystem shall be capable of reconfiguration to remove and add optional communication links and ISS subsystems, control of different types of communication interface hardware, and controlling different types of communication configurations (Star, Ring, and Bus networks) simultaneously.

The physical layout of the ISSes plus nodes had to be addressed. Appendix B gives the floor layout diagrams for the Engineering Branch Laboratory with ISS and node locations identified. The maximum ISS-to-node separation is 200 feet. The maximum node-to-node separation is 300 feet since the Engineering Branch Laboratory is made up of three floors of equipment. These figures will hold true in future EW laboratory expansion efforts.

The design goal for the network for Meam-Time-To-Restore (MTR) should be determined. MTR instead of Mean-Time-To-Repair (MTTR) is used to indicate a return to an operational state for the network by replacing major subsystems of the network. All software must be thoroughly documented and the hardware should be designed for easy modular replacement.

As a final constraint on EWNET, network reliability will be addressed in the design. History generally indicates there is improved reliability where design goals are established prior to development versus where reliability is a fallout of the system design. The following are realistic design goals which address reliability (Ref. 10:15):

1. Failure of a particular node will not render the network inoperative.

2. Optional or alternative interconnection paths between nodes can be established and assured if the primary or direct path is inoperative.

3. Stored data will be preserved in spite of electrical failure.

Summary

Due to the results obtained through the use of the user survey it was possible to specify a comprehensive list of functional requirements and estimate their relative importance. The aspects of resource sharing was the most important use that the users wished to see implemented first. User-interface and security were considered the most important user-oriented functional requirements, while flexibility was considered the most important design-oriented functional requirement. Finally, EMI, communication subsystem, and physical layout considerations were

accessed. Appendix A contains a complete compilation of the user survey results.

Chapter III documents the translation of the general functional requirements identified by this chapter into more detailed system specifications.

III. EWNET System Requirements

Introduction

This chapter translates the general functional requirements identified in the last chapter into more detailed system specifications. The first section of this chapter addresses the system hardware requirements while the second section specifies the system software requirements. The hardware system requirements addressed are the network topology, the host computers to be included in the network, the selection of a suitable node computer for the network and the choice of an appropriate transmission medium for the EWNET communication links. The system software requirements are specified using DeMarco's Structured Analysis technique (Ref. 9). A description of the components of the technique is given and its use is justified. Then, the structuring technique used to arrive at the proposed protocol hierarchies is described (Ref. 11:10-21). Using the Structured Analysis techniques described, the system software requirements are specified at the defined protocol hierarchic levels. Finally, the physical protocol specifications are stated.

The requirements specification was actually an iterative process. While some of the specifications found in this chapter could be derived directly from the functional requirements, many actually followed from design decisions made in the following chapter. These requirements would then trigger new design decisions which might in turn modify the

requirements specification further. What follows is the final results of this iterative process.

System Hardware Constraints

The functional requirements of Chapter II were used to derive the following detailed specifications of the hardware that was required for EWNET. This was done by considering how the functional requirements of the system place constraints on the system hardware. These constraints were then used to derive the more detailed hardware specifications.

Topology. The primary requirement for the topology was that the topology had to be flexible, and must be easy to expand through the addition of more hosts and nodes. Additionally, since availability was of major concern, the topology had to contain built-in redundancy which would not be degraded when one of the host (ISS) computers decided to drop out of the network (voluntarily or involuntarily). The topology also must not contain bottlenecks that will limit the throughput on the network below the stated level or that will increase the response time to an unacceptable high level due to queueing delays. The response time requirement may also impact the topology by limiting the number of nodes between any two host computers since the combined queueing delays may increase the response time to an unacceptable level.

Host Computers. The requirements for the host (ISS) computers to be included in EWNET was directly related to their need to access centrally located databases, their level of standardization, and their

peripheral power. EWNET must meet all standardization requirements presently defined within the Engineering Branch Laboratory. Therefore, the level of host computer sophistication and standardization must be as defined in the Standardized Integration Support Station System Specification" (Ref. 3). Finally, the usefulness of the network would be enhanced, if those host computers used most often in the Engineering Branch Laboratory were included in EWNET.

Nodes. The requirements for the nodes in the network also were addressed. Since the host (ISS) computers must be able to drop out of the network at anytime without degrading the network capabilities, the host (ISS) computers can not act as the node computers. Therefore, all node computers must be individual stand-alone computers that meet all the standardization requirements defined in the "Standardized Integration Support Station System Specification" (Ref. 3). Because EWNET is to be an off-the-shelf standardized local computer network, all centrally located databases and backup ISS functions should be hosted on a Standard Engineering Branch Computer (Ref. 3:16-21). Therefore, it should be possible to combine the Engineering Branch Installation Host Computer function with the EWNET Node Computer function, given a sufficiently sophisticated computer. This implys that the node computers must be capable of handling lower-level network protocols. The nodes should contain their own set of peropherals to provide backup peripheral capability for the ISSes plus to be used when accessing the node computer directly. Furthermore, the node should have the capability to collect performance monitoring statistics. Finally, a

compiler should be available for the node CPU so that the protocol software may be written in a higher order language. This would reduce the magnitude of the implementation effort and enhance the maintainability of the software.

Transmission Medium. There were several system requirements for the transmission medium. First, it must support the data transmission rates necessary to meet the throughput and response time requirements. Second, it must provide reliable communication links to avoid degrading throughput and response times. Otherwise, if a high percentage of blocks of data required retransmission then both response time and throughput would suffer. The same is true when forward error correction is used and a high degree of redundancy is required (Ref. 12:202-208). However, the primary constraint on the transmission medium is that it must provide secure communications in conjunction with a high noise immunity. Finally, the transmission medium should be easily re-routed to allow the topology to be reconfigured with network growth.

Software Specification Tools

Introduction. While the user surveys provided an excellent tool for specifying the functional requirements for EWNET, and some of the more hardware-related system requirements, a technique was required to translate the EWNET functional requirements into system software requirements. Techniques available for this task included DeMarco's Structured Analysis Technique, SofTech's Structured Analysis and Design

Technique (SADT), IBM's Hierarchical Input-Process-Output (HIPO) diagrams, and various problem statement languages (Ref. 9). DeMarco's Structured Analysis technique was chosen since it offered several advantages over the other methods. But before these advantages can be understood, it will be necessary to understand the basic components of the Structured Analysis technique.

Structured Analysis. Structured Analysis is a technique using data flow diagrams (DFDs) and a data dictionary which uses Structured English, decision tables, and decision trees to describe the DFDs. The data flow diagram ia a graphic tool used to depict the logical flow of data through a program or a system. The basic components of a DFD are shown in Figure 2, page 35. The first component is the data flow. It is a "pipeline" of other data flows and of data elements. The data elements are the basic data types that can not be partitioned further and still retain their meaning. The data flow / elements are identified as the labeled arrows that connect the circles on the DFDs. The circles on the DFDs are known as transforms (each identifying a function that transforms data). Transforms convert input data flows to output data flows and is the second component of the DFDs. The boxes represent any mechanism by which information enters or leaves the system. Finally, files are the last component and are repositories of information within the system. They are depicted by stright lines. Access to and response from a file is depicted by a pair of unlabeled arrows. The arrow to the file represents the search argument. The arrow from the file represents retrieved information or status. The DFDs are layered starting with an

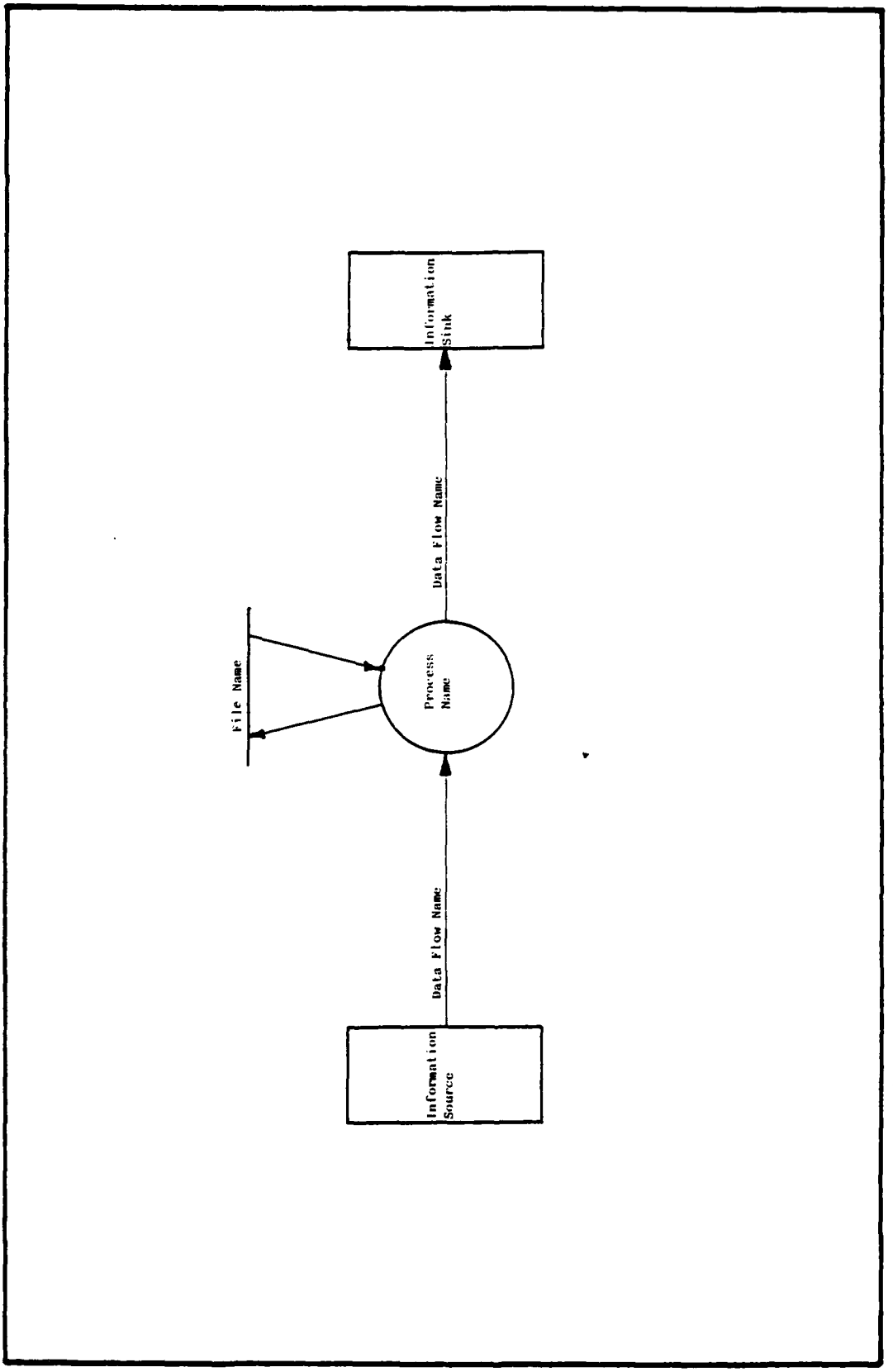


Figure 2 DFD Components

overview DFD. The overview DFD is used to show on one page the major processes required in a system. These processes normally must be broken down to show the detailed processing requirements within each major function. To understand the detail of a given process you should refer to a DFD with the same numerical prefex as the process in question. Each process on the "parent" diagram is a consolidation of the network shown on the "child" diagram. The partitioning of the processes continues until the data flows entering and leaving a process consist of only one data element each. At this point, a process description is written for the process and the process is not expanded into a lower-level diagram. The process descriptions, data flow and element descriptions, and file descriptions for all the DFDs are compiled into the data dictionary.

Using the above "working" definition of Structured Analysis, it is now possible to examine the advantages that Structured Analysis offers. First, it was based upon the concept of partitioning. This facilitates Top-Down analysis, as major user functions are decomposed into their subfunctions. Thus, allowing the large and complex EWNET requirements problem to be approached in an orderly manner rather than causing the engineer to be overwhelmed by the vast amount of requirements to be specified. Second, the data dictionary focuses on data flows rather than on processes, resulting in clearly defined interfaces. Third, the process definitions could be easily specified using Structured English, a tool incorporated into Structured Analysis. Fourth, redundancy in the Structured Specification is eliminated due to the organization

associated with the DFD / data dictionary combination. This made the task of maintaining the specification and changing it much easier. Fifth, the data flow diagrams were a two-dimensional presentation of the requirements, thus presenting the structure in a much clearer manner than would otherwise be possible using the traditional linear and voluminous specification approach. Sixth, Structured Analysis differentiated between the logical and the physical envıroment, thus allowing the functional software requirements to be specified without being concerned about the actual hardware that would execute those requirements. Finally, it was easier to spot inconsistencies and gaps in the structured specification and to correct those deficiencies. The major disadvantage was that it was time comsuming to generate the DFDs.

Protocol Hierarchies

Introduction. now that we have discussed the need for using Structured Analysis in the specification phase, it becomes apparent that a method would be required whereby the different software protocol functions could be further layered to facilitate easier implementation. In keeping with standardization efforts (worldwide), it was decided to layer the protocols for EWNET using the International Standards Organization (ISO) model for network protocols (Ref. 11:15-16). The Reference model of Open Systems Interconnection (OSI), as ISO calls it, has seven layers as shown in Figure 3, page 38 (Ref. 11:16). They are the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the

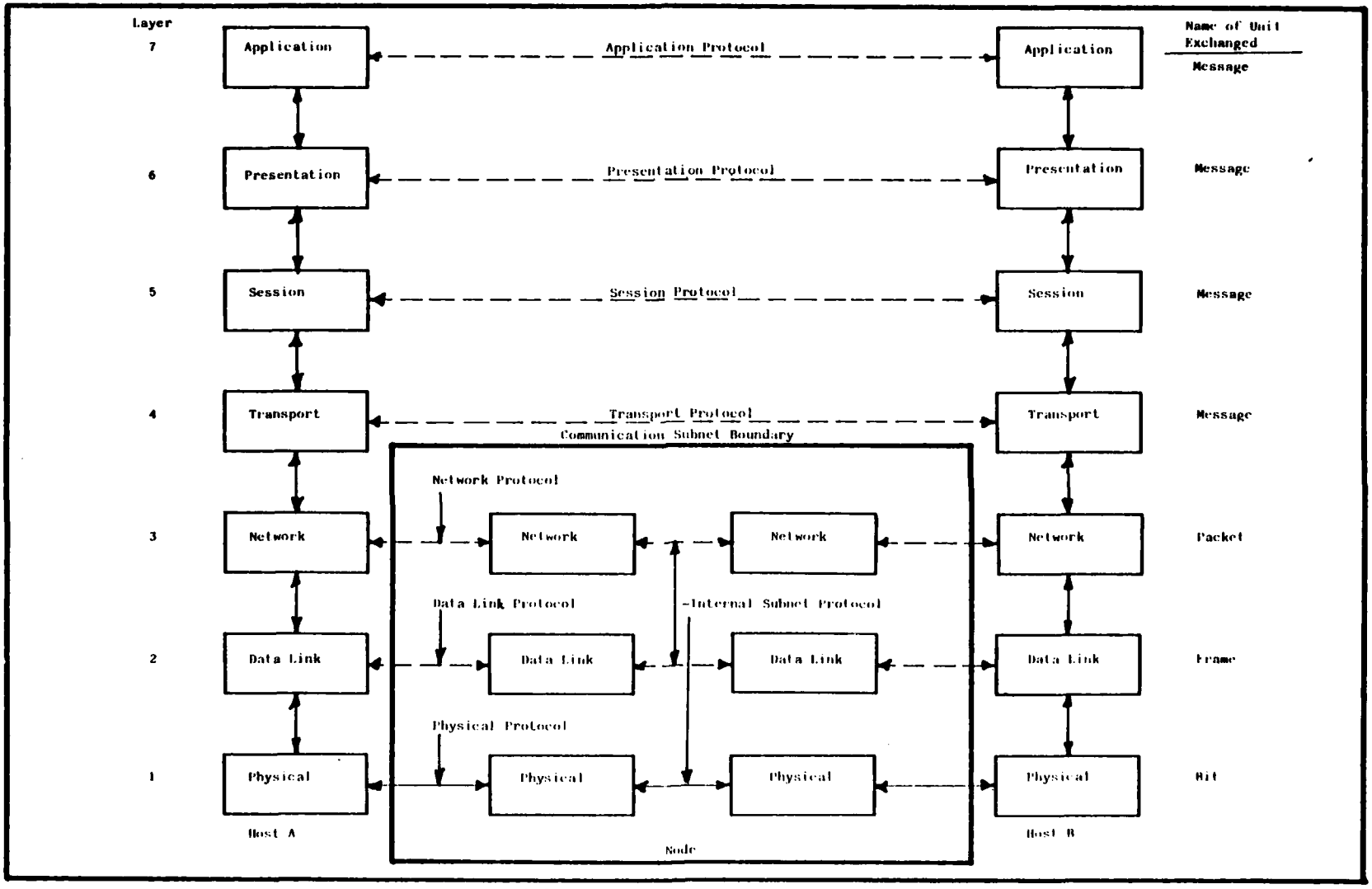


Figure 3 ISO OSI Network Architecture Model

application layer.

Physical Layer. This layer is concerned with transmitting raw bits over a communication channel.

Data Link Layer. The task of the data link layer is to take a raw transmission facility and transform it into a link that appears free of transmission errors to the network layer. It accomplishes this task by breaking the input data up into data frames, transmitting the frames sequentially, and processing the acknowledge frames sent back by the receiver.

Network Layer. This layer determines the chief characteristics of the host-to-node computer interface, and how packets, the units of information exchange in layer 3, are routed within the subnet.

Transport Layer. The basic function of this layer, is to accept data from the session layer, split it up into smaller units, pass these to the network layer, and ensure that the pieces all arrive correctly at the other end.

Session Layer. The session layer is the user's interface into the network. It is with this layer that the user must negotiate to establish a connection with a process on another machine.

Presentation Layer. The presentation layer performs functions that are requested sufficiently often to warrent finding a general solution for them, rather than letting each user solve the problems.

Application Layer. This layer is used to control remote file access, network access, and to manage any statistical functions on the network.

Finally, it should be noted that the session lager, presentation layer, and the application layer are combined in several different ways to form one or more high-level protocol layers not discussed here. This is done depending on the individual system requirements.

Structured Specification of the Protocol Requirements

Context Diagram. The context data flow diagram shows the system boundary and interface with the user. From Figure 4, page 41, it can be seen that the network operating system must include everything between the user input from a keyboard that is connected to a computer in the network, to the response that the user receives at his display. Thus the Network Operating System includes all the computer operating systems in the network as well as all interface software and hardware required to implement the network functional requirements. To initialize the network, a configuration bootstrap process is also required. Table 4 shows the layers of protocol that are defined in the set of DFDs that follow. Table 5 describes the process hierarchy for the first set of DFDs, the Network Operating System.

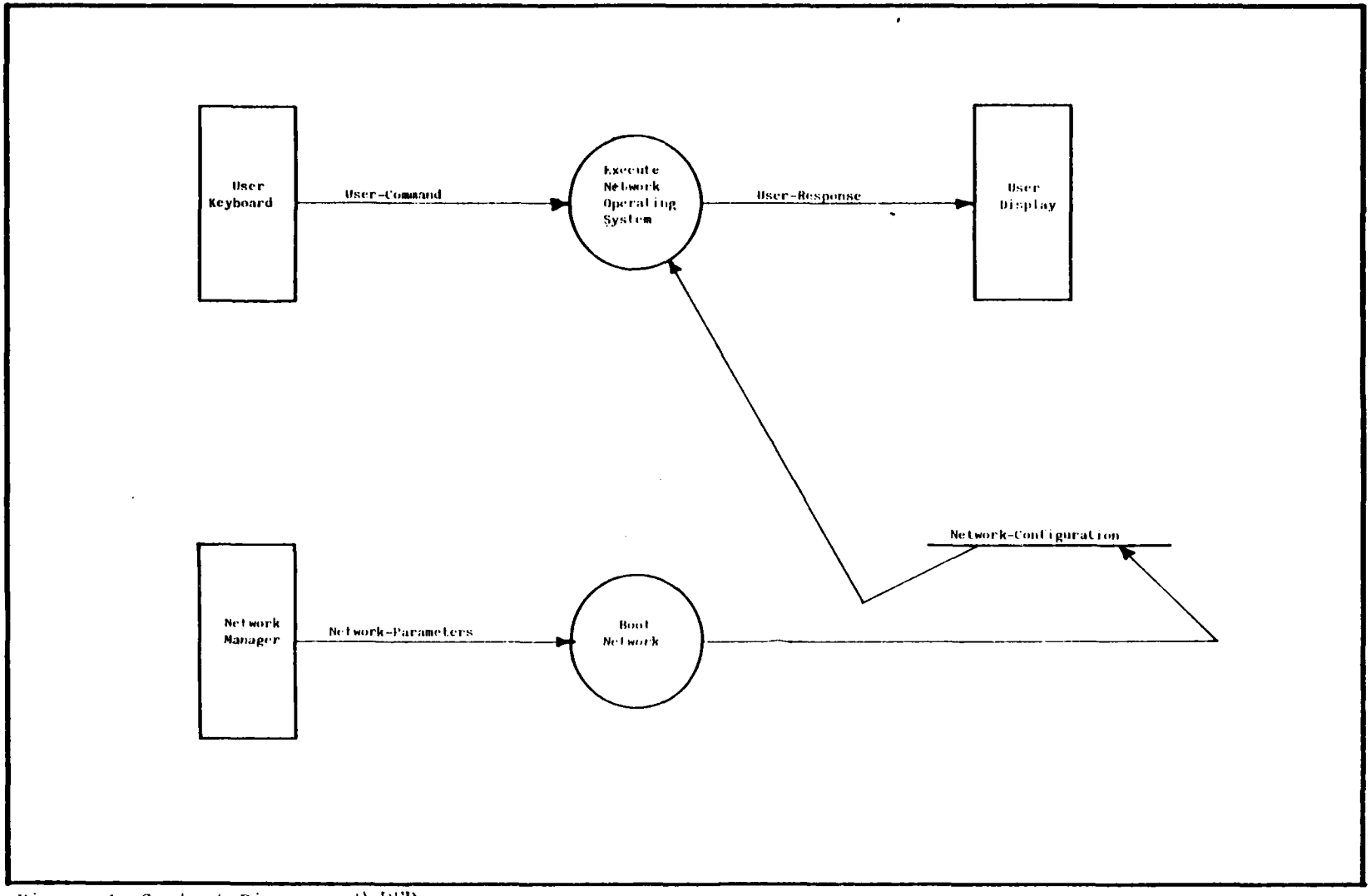


Figure 4 Context Diagram 0 DFD

Table 4
Protocol Layers

EWNET		ISO - OSI
NETWORK OPERATING SYSTEM	----------	ARCHITECTURE PROTOCOL
	----------	PRESENTATION PROTOCOL
	----------	SESSION PROTOCOL
ARCHITECTURE PROTOCOL (DECNET)	----------	ARCHITECTURE PROTOCOL
	----------	PRESENTATION PROTOCOL
	----------	SESSION PROTOCOL
TRANSPORT PROTOCOL (DECNET)	----------	SESSION PROTOCOL
	----------	TRANSPORT PROTOCOL
NETWORK PROTOCOL	----------	NETWORK PROTOCOL
DATA LINK PROTOCOL (HDLC)	----------	DATA LINK PROTOCOL
PHYSICAL PROTOCOL (RS-232-C)	----------	PHYSICAL PROTOCOL

Table 5---Network Operating System Process Hierarchy

1.0 Execute EWNET Protocol at Primary Node

1.1 Determine Command Type

1.2 Execute Help Commands

1.2.1 Decode Help Command

1.2.2 Provide General Network Info

1.2.3 Provide Procedure for Transferring Files

1.2.4 Provide List of Active Hosts and Devices

1.2.5 Provide Network Startup Procedures

1.2.6 Code Help Responses

1.3 Execute Architecture Level Protocol

1.4 Execute Transport Level Protocol

1.5 Execute Network Level Protocol

1.6 Execute Link Level Protocol

1.7 Execute Physical Level Protocol

1.8 Decode Protocol Responses

2.0 Execute EWNET Protocol at Secondary Node

System Diagram. The system diagram translates the key functional requirements into the software requirements for the network operating system as shown in Figures 5 and 6, pages 44 and 45. First, the user command must be examined to determine whether it is a help command, network command, or local command (1.1). If it is a help command, then the appropriate help information must be output to the user. Depending upon the Type of help command, this process may require access to the Dialogue Process Table (1.2).

If the incoming command is a network command, then send it to the correct protocol level to be implemented. If the command was to transfer a file, then the Architecture Level Protocol will execute the command (1.3). Else, if the command is to start or stop the network then the Transport Level Protocol will execute the command (1.4). Once the Architecture Level Protocol has been implemented for a file access or transfer then the file plus the Architecture Protocol Header will be sent to the Transport Level Protocol (1.4). The Transport Protocol will determine if the file transfer or initialization request is for a satellite node or a remote node. If the data is to go to a remote node then the Transport Protocol Header is added to the packet and it is sent to the Network Level Protocol for processing (1.5). The Network Level Protocol adds routing information headers and sends it to the Data Link Level Protocol (1.6). Also, if the data to the Transport Level Protocol

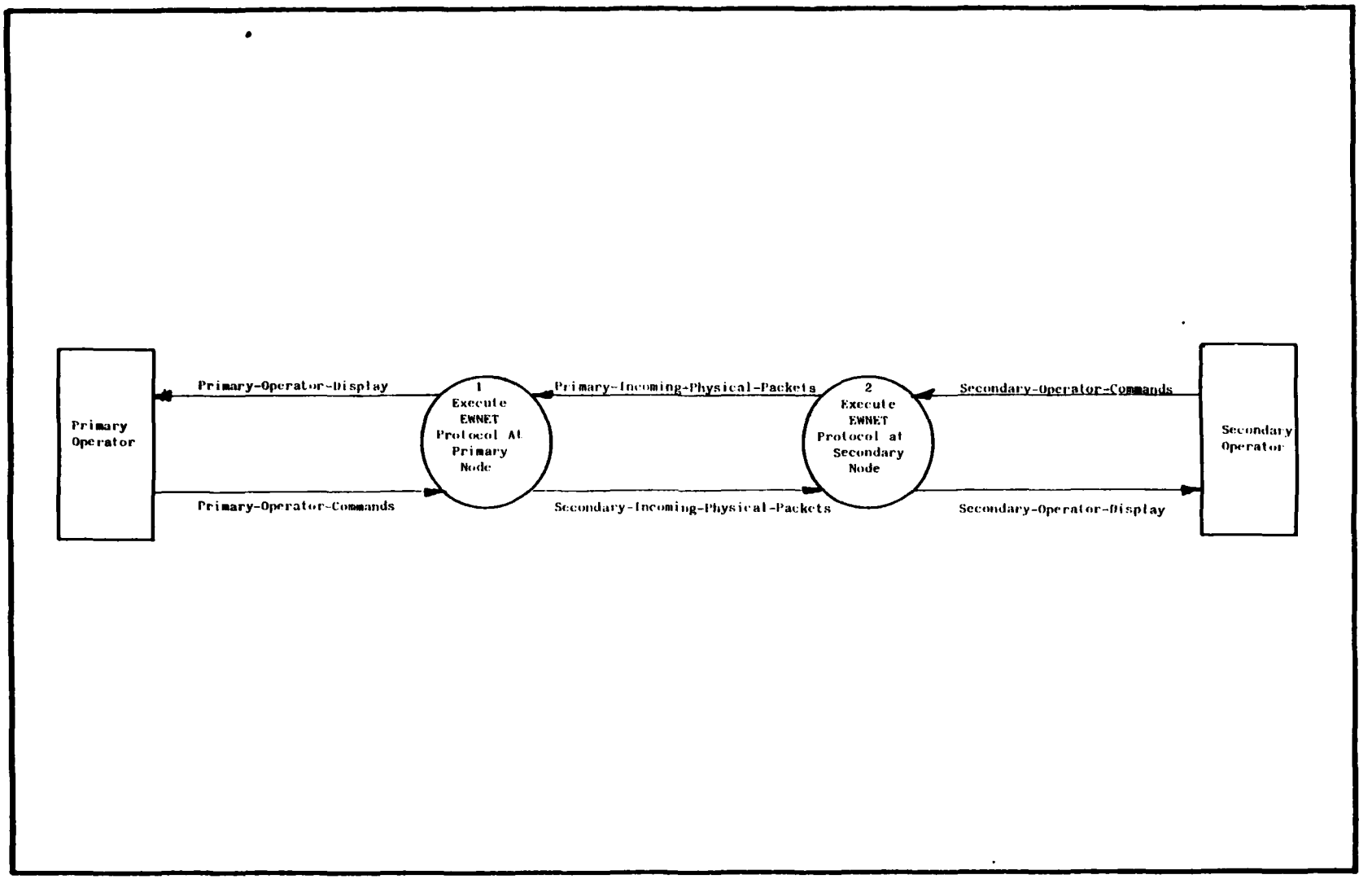


Figure 5 Ewnet Overview (Network Operating System) DFD (0.0)

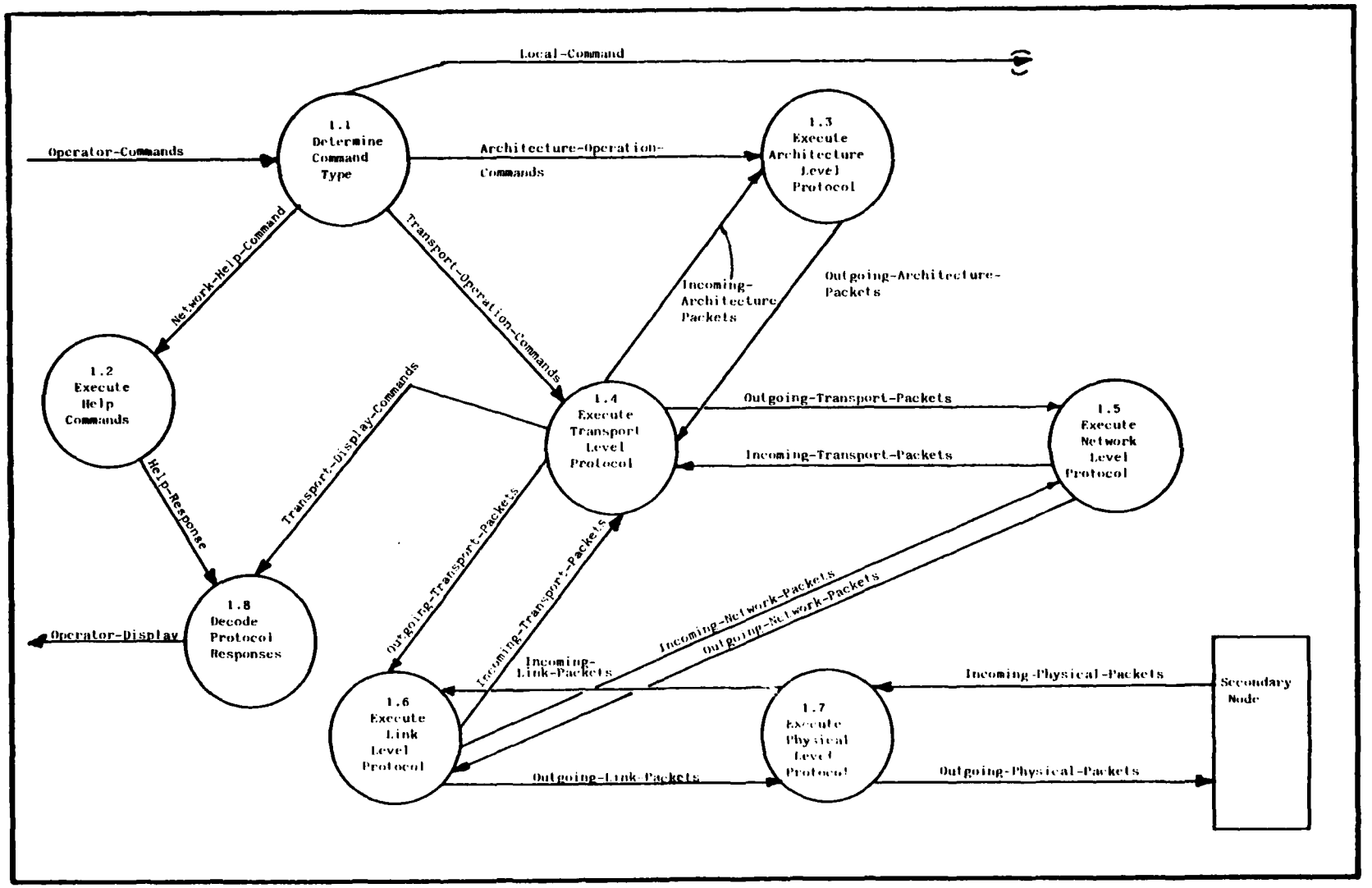


Figure 6 Execute Ewnet Protocol at Primary Node (1) DFD

is to be sent to a satellite node then send the data with the Transport Level Protocol Header directly to the Data Link Level Protocol (1.6). The Data Link Level Protocol then adds its own header and sends the entire packet to the Physical Level Protocol (1.7), which transmits the bit stream. Finally, the response that has been outputed from one of the three processes must be transmitted back to the user as the network response (1.8).

If the user command was a local command, then the command must be routed internally to perform the required function. In all processes listed in Figure 6, page 45, there is the possibility for errors to develop, and thus, error messages must be generated in place of the expected result.

Help User Requirements. The user help process may be specified further by the lower-level DFD shown in Figure 7, page 47. The transmitted help request must be classified as either a general information request, a file transfer information request, a list of the active host and device names, or a protocol start information request (1.2.1). If it is a general information request, then a menu selection consisting of the available commands and their formats must be output along with the formates for the more specific help requests (1.2.2). If the transmitted help request is a file transfer information request, then format for the transfer file commands must be output (1.2.3). If the transmitted help request is a list of the active hosts and device names request, then this list must be output by accessing the Dialogue Process Table (1.2.4). If the transmitted help request is a protocol

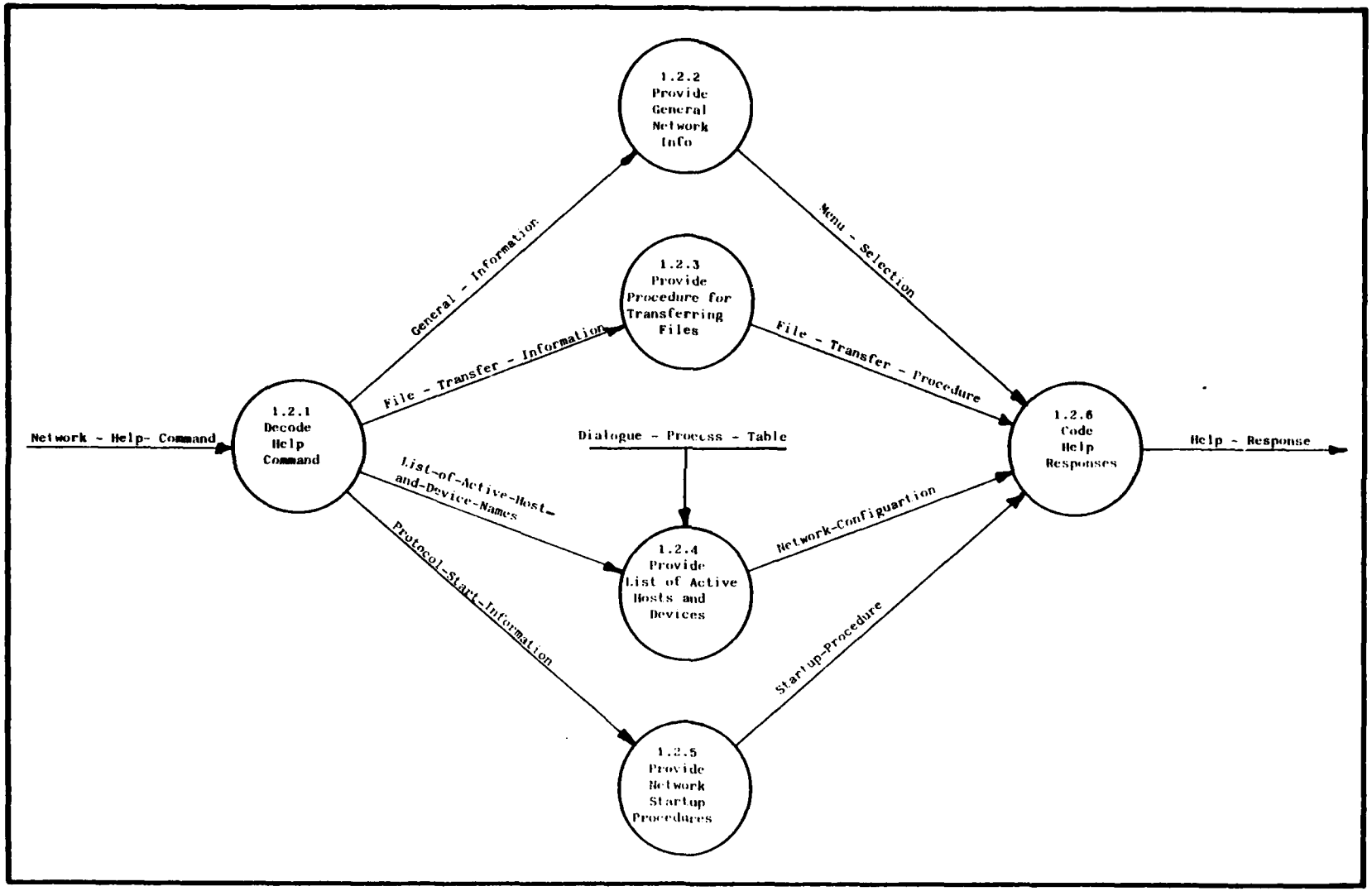


Figure 7 Execute Help Commands (1.2) DFD

start information request, then the commands necessary to start, stop, and abort the session must be output (1.2.5). Finally, all help responses are output to the user as a help response (1.2.6).

Architecture Level Protocol Requirements. The previous protocol requirements were specified to implement the applications of obtaining user help information and identifing network commands. The identifing of the network commands process relies upon an architecture file transfer mechanism which was treated as a primitive by this process. If the DFD partitioning process had been followed as specified by DeMarco, then the partitioning of the file transfer mechanism would have been duplicated several times. However, by defining the file transfer mechanisms as primitives that are used by several processes, it was possible to start a new set of DFDs for this mechanism without having to duplicate them for each process. Table 6 shows the overall process hierarchy for the Architecture Level Protocol DFDs. In the interest of conservation of material, the detailed DFDs for the Architecture Level are not included in the text, but are included in Appendix C, as are all upper and lower-level DFDs for the entire software specification.

Table 6---Architecture Level Protocol Process Hierarchy

1.0 Execute Architecture Protocol at Primary Node

1.1 Decode Architecture Packets

1.2 Decode Status Packets

1.2.1 Check for Errors

1.2.2 Decode MACCODE Field

1.2.3 Decode Unsupported MICCODE Field

1.2.4 Decode Pending MICCODE Field

1.2.5 Decode Format MICCODE Field

1.2.6 Decode File MICCODE Field

1.2.7 Decode Sync MICCODE Field

1.2.8 Decode Successful MICCODE Field

1.3 Generate Control Packets

1.3.1 Decode Control Type

1.3.2 Generate Control Connect Packet

1.3.3 Generate Control Seq-Get Packet

1.3.4 Generate Control Seq-Put Packet

1.3.5 Generate Control Key-Get Packet

1.3.6 Generate Control Key-Put Packet

1.3.7 Generate Control Add-Get Packet

1.3.8 Generate Control Add-Put Packet

1.3.9 Code Get Fields

1.3.10 Code Put Fields

1.3.11 Code Control Fields

1.4 Execute Startup Packets

1.4.1 Generate Configuration Packet

1.4.2 Check for Configuration Errors

1.4.3 Decode Configuration Fields

1.4.4 Generate Attributes/Access Packet

1.4.5 Generate Access/Erase/Rename Packet

1.4.6 Generate Access/ECF/SCF Packet

1.4.7 Generate Attributes Packet

1.4.8 Check for Acknowledge Errors

1.4.9 Check for Attributes Errors

1.4.10 Check for Access Errors

1.4.11 Decode Attributes Fields

1.4.12 Code Setup Errors

1.4.13 Code Setup Packets

1.5 Execute ACC/ACK Packets

1.5.1 Generate Access Packet

1.5.2 Decode Access Fields

1.5.3 Generate Access Complete Packet

1.5.4 Generate Acknowledge Packet

1.5.5 Check for Access Complete Errors

1.5.6 Decode CMPFUNC Fields

1.5.7 Generate ATTRIB/ACK Packet

1.6 Execute Control Packets

1.6.1 Decode Control Packet

1.6.2 Decode Connect Fields

1.6.3 Decode Get Fields

1.6.4 Decode Put Fields

1.7 Execute Continue Packets

1.7.1 Decide on Next Action

1.7.2 Decide on Required Action

1.7.3 Generate Continue Abort Packet

1.7.4 Generate Continue Skip Packet

1.7.5 Generate Continue Only Packet

1.7.6 Code Continue Packets

1.7.7 Generate Data Packet

1.7.8 Code Continue Errors

1.8 Code Architecture Packets

1.8.1 Start/Stop Timer

1.8.2 Code Working Packets

1.8.3 Generate Status Packets

1.8.4 Terminate Logical Link

1.8.5 Terminate Data Stream

1.8.6 Code File Packets

1.9 Check for Continue Error

2.0 Execute Architecture Protocol at Secondary Node

The Architecture Layer provides the network functions for the user layer. Modules in this layer include network remote file access modules, a remote file transfer utility, and a remote system loader module.

The use of the network remote file access modules and the remote file transfer utility are limited to the following type of files:

1. Sequential - Each record's position depends on the position

of the previous record. Records may not be processed in any other order (Ref. 13:298).

2. Relative - Each record in the file has a unique identifying number, its record number. Records may be accessed randomly by specifying their record number in a control message (Ref. 13:298-299).

3. Direct/Indexed - These files have records organized according to some classification method, usually an access key. Within a particular key, the records are assumed to be sequential (Ref. 13:299).

The reason for this is because it was felt that these three types of files were the types projected to be most prevalent on the network. Figure 8, page 53, is the overview DFD for the Architecture Level Protocol. Obviously, the most important function of the Architecture Protocol is to transfer data to and from I/O devices and mass storage files independent of the I/O structure of the system being accessed. This implys that the protocol must be transparent to the files being transferred. From the network, it must appear as if the EWNET systems support the Architecture Protocol Messages directly within their file systems. The protocol must set up the conversation path for remote file access, transferring data over the link and terminating the logical link. The EWNET processes will implement the Architecture Protocol using the Transport/Network Protocol and the network facilities for the creation of the logical link and for flow control. Figure 9, page 54,

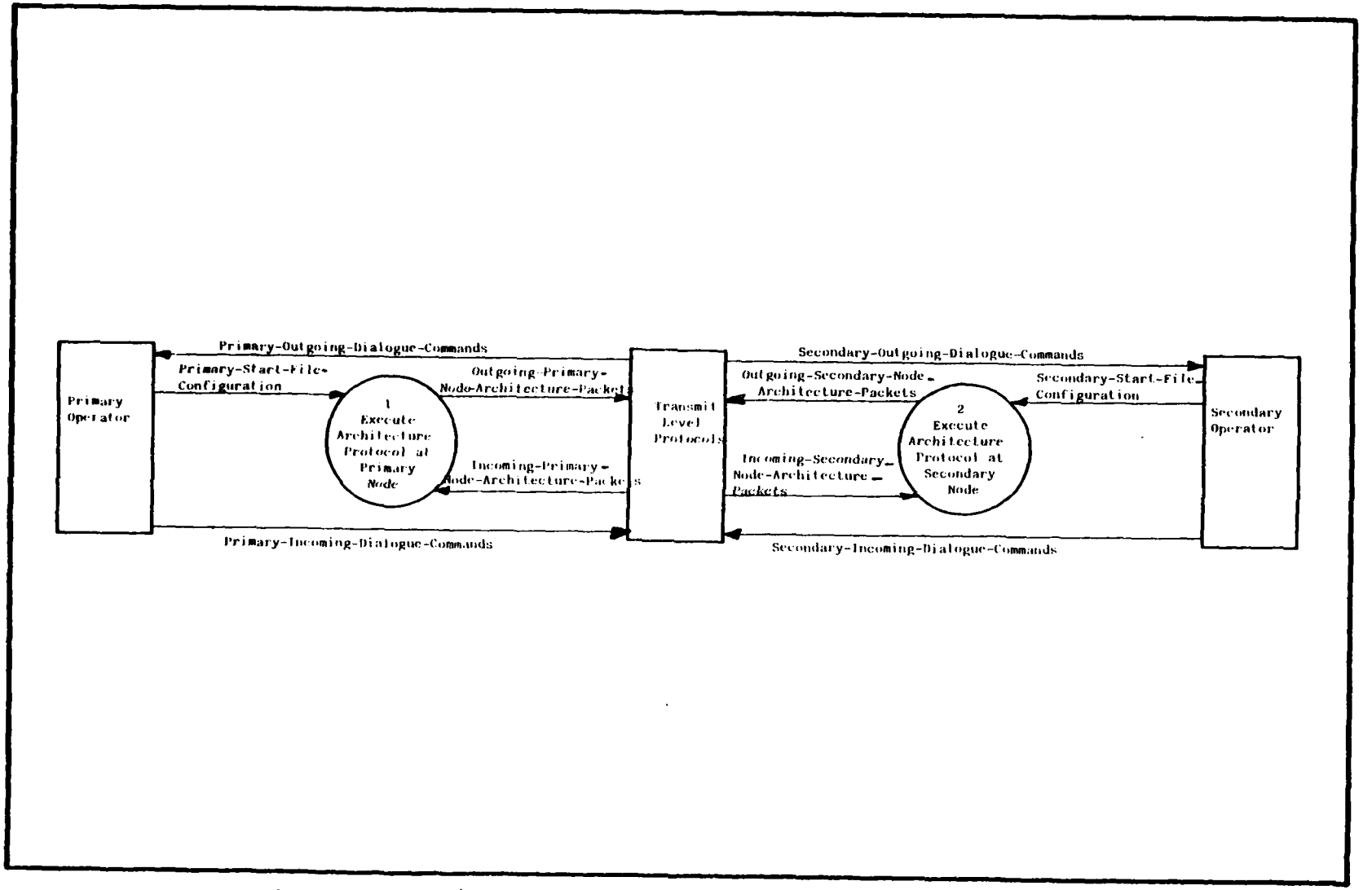


Figure 8 Ewnet (Architecture) Overview DFD

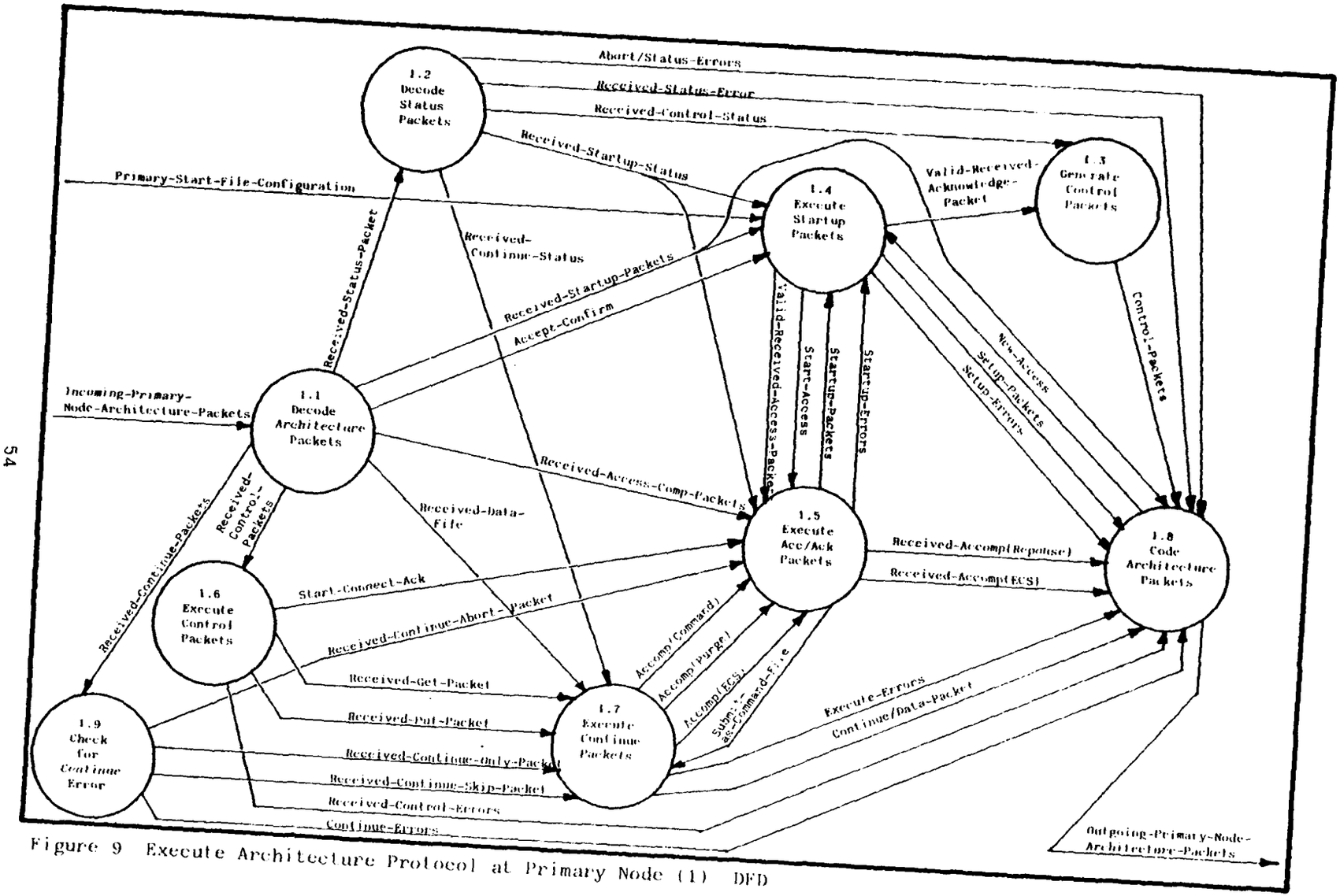


Figure 9 Execute Architecture Protocol at Primary Node (1) DFD

is the upper-level DFD for the Architecture Level Protocol.

Once the link is established by the Transport Level Protocol, the EWNET processes the exchange of Architecture Level messages to set up the file access. These initial set up messages include configuration, file, data mode, and format information. A data stream is then set up to transfer data over the link. Data may be sent either to the accessing node as in a retrieve operation or from it as in a store operation. One control message sets up the data stream for both sequential and random access file transfer. After file transfer is accomplished, additional Architecture Level messages terminate the data stream. A disconnect request terminates the logical link. Status messages are returned if there are errors in the set up procedure, file transfer, or termination.

Transport Level Protocol Requirements. The previous protocol requirements were specified to implement the applications of file transfers on the network. These file transfer processes rely upon a Transport Protocol (transferring mechanism) which was treated as a primitive by these processes. Since the Transport Protocol is a primitive that is used by several processes, it was possible, as with the architecture protocol, to start a new set of DFDs for this mechanism without having to duplicate them for each process. Table 7 shows the overall process hierarchy for this set of DFDs. As with the Architecture Level Protocol DFDs, only the upper-level DFDs will be included in the text explaination for this protocol.

Table 7---Transport Level Protocol Process Hierarchy

1.0 Execute Transport Protocol at Primary Node

1.1 Decode Route Header

1.1.1 Decode RTHDR Field

1.1.2 Decode MSGFLG Field

1.1.3 Generate First Disconnect Confirm Packet

1.1.4 Examine Logical-Link Database

1.1.5 Pass to Satellite

1.2 Decode Packet Type

1.3 Execute Incoming Dialogue Message

1.3.1 Execute Dialogue Segment

1.3.1.1 Decode Dialogue Message

1.3.1.2 Examine Adjacent Node Parameter and Decode

1.3.1.3 Break Dialogue Data into Segments

1.3.1.4 Examine Data Flow Control Parameters

1.3.1.5 Piggyback Data Acknowledge

1.3.1.6 Assign Segment Number Mod 4096

1.3.1.7 Load and Delete Data Memory

1.3.1.8 Check Data Retransmit

1.3.1.9 Code Data and I/L Packets

1.3.2 Execute I/L Packet

1.3.2.1 Examine Interrupt Flow Control Parameters

1.3.2.2 Generate Interrupt-Link Services Packet

1.3.2.3 Piggyback I/L Acknowledge

1.3.2.4 Code I/L Packet

1.3.2.5 Generate Data-Link Services Packet

1.3.2.6 Assign Packet Number Mod 4096

1.3.2.7 Load and Delete I/L Memory

1.4 Execute Data Packet

1.4.1 Determine Data Packet Type

1.4.2 Decode Data ACKNUM Field

1.4.3 Decode Data Segment Number

1.4.4 Decode I/L Segment Number

1.4.5 Decode I/L ACKNUMI Field

1.4.6 Generate Data Acknowledge Packet

1.4.7 Generate Data Negative Acknowledge Packet

1.4.8 Load Receive Buffer until Full or LS

1.4.9 Generate I/L Negative Acknowledge Packet

1.4.10 Generate I/L Acknowledge Packet

1.4.11 Decode Valid I/L Packet

1.4.12 Code Acknowledge Packets

1.4.13 Code Data Packets

1.4.14 Decode Link Services Packet

1.5 Execute Startup Packet

1.5.1 Transition Link to On-Line Mode

1.5.2 Decode Functions Field

1.5.3 Decode Initialization Packet

1.5.4 Generate Node Initialization Packet Verify = 0

1.5.5 Decode Remaining Fields

1.5.6 Generate Node Verification Packet

1.5.7 Generate Node Initialization Packet Verify = 1

1.5.8 Decode Password

1.5.9 Code Initialization Packets

1.6 Execute Control Packet

1.6.1 Execute Connect Packet

1.6.1.1 Generate Connect Initiate Packet

1.6.1.2 Decode Control Packet

1.6.1.3 Decode Connect Initiate Packet

1.6.1.4 Dialogue Process External End

1.6.1.5 Decode Connect Confirm Packet

1.6.1.6 Generate Connect Confirm Packet

1.6.1.7 Generate Disconnect Initiate Packet

1.6.1.8 Code Control Packet

1.6.2 Execute Disconnect Packet

1.6.2.1 Decode Disconnect Confirm Packet

1.6.2.2 Decode Disconnect Initiate Packet

1.6.2.3 Dialogue Process End

1.6.2.4 Generate Disconnect Confirm Packet

1.7 Execute Acknowledge Packet

1.7.1 Determine Acknowledge Type

1.7.2 Decode Data ACK Packet

1.7.3 Decode I/L - ACK Packet

1.8 Execute Outgoing Transport Packets

1.8.1 Examine DSTADDR Field

1.8.2 Add RTHDR Field

1.8.3 Is Routing Necessary and Present

1.8.4 Pass to Correct Adjacent Node

1.9 Code Outgoing Dialogue Message

1.10 Send to Network Protocol

4.0 Execute Transport Protocol at Secondary Node

The overview DFD for the Transport Layer is shown in Figure 10, page 60. Obviously, the most important function of this protocol is to transmit data from the source host to the destination host. To accomplish this function the Transport Protocol must retain primary responsibility for:

1. Establishing, maintaining, and destroying communication links between different user and network application program modules.

2. Performing initialization between adjacent nodes.

3. Acknowledgement of all messages.

4. Detecting node and line failures.

In addition, the Transport Protocol will provide the facility for two dialogue processes (typically, user-written or network application

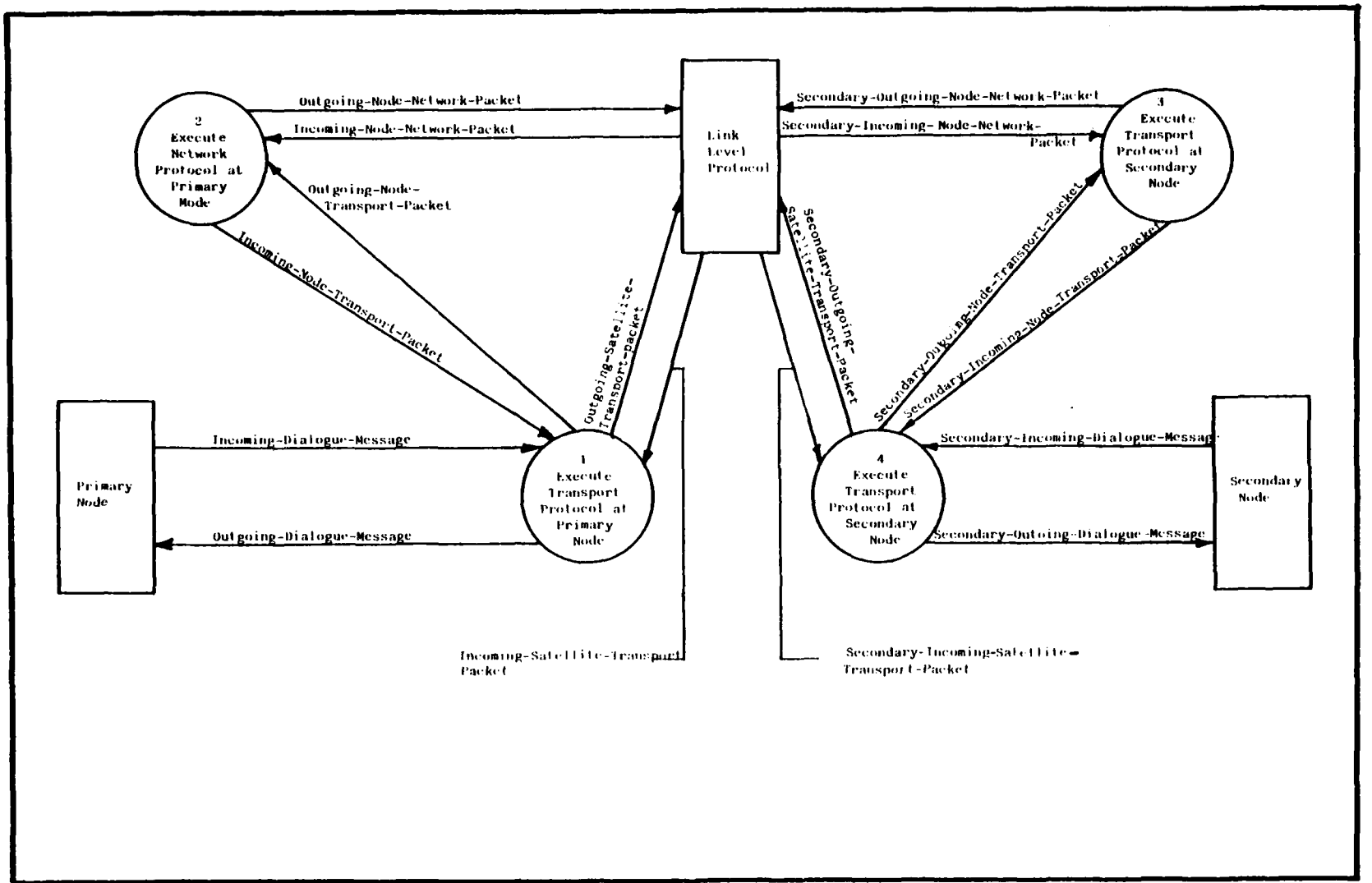


Figure 10 Ewnet (Transport/Network) Overview DFD

modules residing at different nodes) to exchange information regardless of their physical location within a network. This facility is referred to as Logical Link Service (Ref. 12:127).

Logical Link Service allows a dialogue process to establish a connection (called a logical link) to another dialogue process. Once the logical link is established, each dialogue process may send data to the other dialogue process via the logical link. When the dialogue is complete, either dialogue process may request that the logical link be disconnected. A logical link provides both guaranteed delivery (that is, delivery of information to an area of storage accessable to the destination process) and sequentiality.

Other network management activities performed by the Transport Level Protocol are data flow control, node initialization, destination node and process identification, and link failure handling.

To support the above functions the Transport Protocol must have two types of messages that it uses. These are Data Messages and Control Messages. Data Messages are also of two types: normal data messages and interrupt messages. Normal data messages can be either an entire message or a message segment. Interrupt messages carry small amounts of information (high-priority information such as an alarm condition).

Control Messages pass information between the Transport modules. These messages are used for the following purposes:

1. Starting up and initializing nodes,

2. Testing,

3. Establishing, maintaining, and terminating logical link operation, and

4. Controlling the flow of data and interrupt messages on a logical link.

The reason for the interrupt data commands is to allow long transfering files to be interrupted for the transmission of the network control commands. If this were not done, a long file transfer could easily degrade the response time of a network command to an unacceptable level.

This concept of interrupting file transfers requires that the transmission medium be shared. Time Division Multiplexing (TDM) or Frequency Division Multiplexing (FDM) could provide multiple paths between any two hosts as could many topologies (Ref. 12). However, since the throughput requirement might possibly grow greater than the channel capacity between nodes on EWNET, FDM was thought to be the best way to meet present and future response time requirements. Therefore, it was decided that, since a large bandwidth transmission medium will be used on EWNET and an initial high throughput would be required, a Frequency Division Multiplexed Packet Switched Protocol would be appropriate for the Transport Level Protocol. Due to the constraint of developing a network for the Engineering Branch that would meet all internal standardization requirements and be an off-the-shelf item, the

Digital Equipment Corporation's DECNET Protocol was chosen for EWNET. The DECNET Digital Network Architecture (DNA) model and its application to the EWNET will be discussed in Chapter 4 of this thesis. Figure 11, page 64, is the Transport Level Protocol high-level DFD for the EWNET.

The Transport Level Protocol is responsible for establishing and destroying a logical link between dialogue processes. This is accomplished via a set of control messages sent back and forth between Transport modules. Once a logical link has been established, data may be exchanged over the link. Information sent over the link can be: (1) normal data -- that is, data segments or messages or (2) interrupt data. Dialogue messages are usually sent as normal data segments over the link.

Unless the message to be sent is small enough to be transmitted through the network intact, the Transport Protocol will break up the message into segments and transmit each segment individually. To ensure proper data sequentiality and delivery, each Transport module employs a segment acknowledgement scheme. This scheme keeps track of data segments sent and ascertains whether or not retransmission is necessary. At any time during a dialogue exchange, the Transport Protocol will allow either process/party to abort or terminate the conversation. When the Transport Protocol has been properly notified to do so, it will disconnect the logical link.

There are two data streams on a logical link. One stream contains interrupt (for user high-priority information) and link service messages

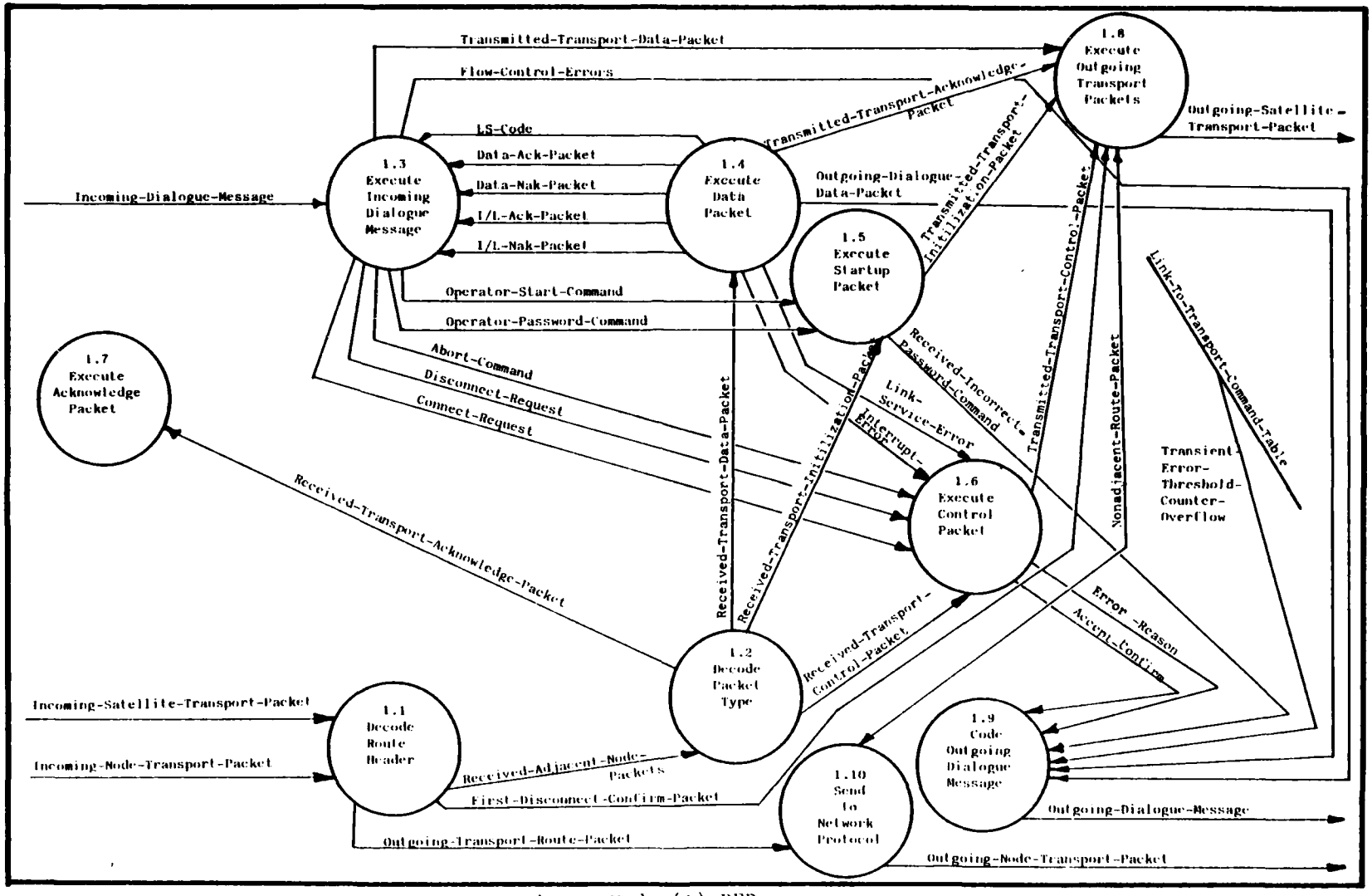


Figure 11 Execute Transport Protocol at Primary Node (1) DFD

(for Transport flow control); the other contains normal data messages. Each stream may be considered a subchannel on the logical link. On the interrupt/link service subchannel, all messages are single Transport segments. On the data subchannel, however, data messages are usually broken up into segments. Transport segment acknowledgement is performed independently for each subchannel.

When normal data (dialogue) messages are broken up into segments, the transmit segment size parameter determines the size of these segments. The value of the parameter is specified when the logical link is established. In addition, transmitter and receiver subchannels must be synchronized for all data message transmission. This is accomplished at the transmitter using the interrupt/link service subchannel transmit number and the data subchannel transmit number and at the receiver using corresponding receive numbers. When a logical link connection is established between two dialogue processes, these numbers are set to 1. As Transport segments are transmitted and acknowledged over the logical link, the appropriate numbers are incremented.

Operating at a level above the segment acknowledgement scheme is the flow control operation. Its purpose is to determine whether there should be a permanent shift from the transmitter to the receiver in the buffering of information. Four parameters are associated with flow control: a data flow control switch ("open" or "closed"); an interrupt request count; a data request switch ("segment"); and a data request count. The receiver may control data flow by sending a link service message that sets appropriate parameters. The receiver can request

interrupt messages, stop data segment flow, allow data segment flow, and request data by means of the segment request count. The transmitter responds appropriately to the link service message of the receiver. When the transmitter has data to send (either normal data or interrupt), it examines the appropriate flow control parameters and request count before sending the data. The receiver may reject segments for which it does not have buffer space available by sending a negative acknowledgement.

Network Level Protocol Requirements. As with the Transport Protocol, the Network Level Protocol is used by several processes and is treated as a primitive by all other protocol levels. Table 8 shows the overall process hierarchy for this set of DFDs.

Table 8---Network Level Protocol Process Hierarchy

2.0 Execute Network Protocol at Primary Node

- 2.1 Decode message type
- 2.2 Check R-Hop Count = 2
- 2.3 Decode Network Header
- 2.4 Increment Hop
- 2.5 Update Routing Table If New
- 2.6 Determine Least Cost Link
- 2.7 Add Network Header
- 2.8 Send Old Values Out Over New Initialized Line
- 2.9 Check N-Hop Count = 2

2.10 Update Network Header

2.11 Every Timer Interval Update Adjacent Line

2.12 Issue Correct Link Cost Over All Links

2.13 Send Packet Over Correct Line(s)

3.0 Execute Network Protocol at Secondary Node

The overview DFD for the network layer is shown in Figure 10, page 60. The most important function of this protocol is to implement what is termed in the communication field a Datagram Service (Ref. 12:60). A Datagram Service delivers packets on a "best effort" basis. That is, the Network Protocol makes no absolute guarantees against packets being lost, duplicated, or delivered out of order. Rather, higher layers of the EWNET Protocol structure provide such guarantees (Transport Level Protocol). The Network Protocol selects routes based on network topology and operator-assigned line costs. The Network Protocol automatically adapts to changes in the network topology, for example, by finding an alternate path if a line or node fails. This protocol does not adapt to traffic loading: the amount of traffic on a channel does not affect the Network Protocol's routing algorithms. Figure 12, page 68, is the Network Level Protocol DFD for the EWNET.

The Network Level Protocol is responsible for providing the following functions:

1. Determines packet paths. A path is the sequence of connected nodes between a source node and a destination node. If

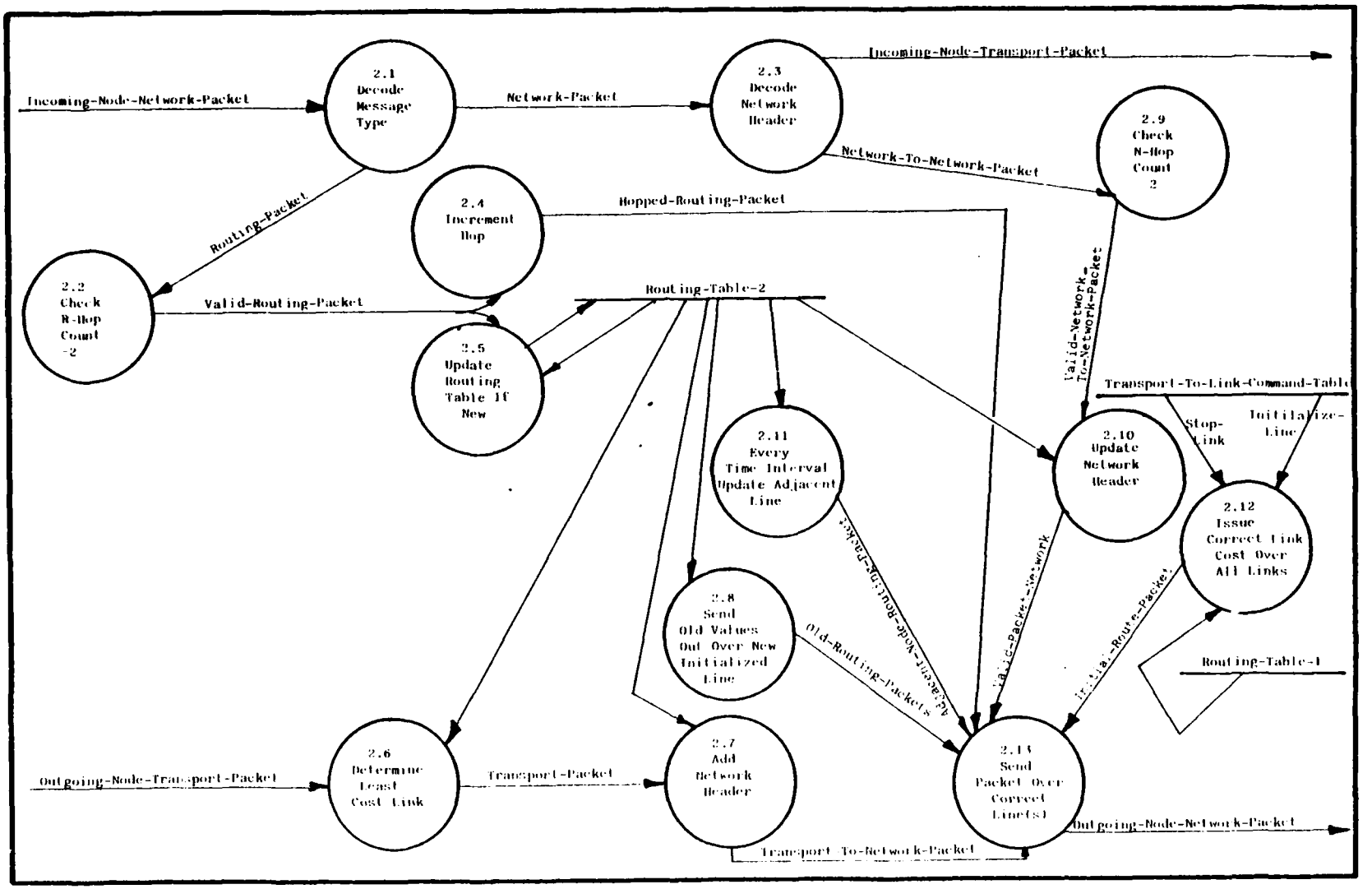


Figure 12 Execute Network Protocol at Primary Node (2) DFD

more than one path exists for a packet, the Network Protocol determines the best path.

2. Forward packets. If a packet is addressed to the local node, the protocol forwards it up to the Transport Level Protocol for disposition. If a packet is addressed to a remote node, the protocol forwards it on to the next line in the path.

3. Manages the characteristics of packet paths. If a line or node fails on a path, the Network Protocol finds an alternate path, if one exists.

4. Periodically updates other Network Level modules. Other Network Level modules are periodically updated so that all nodes in the network are aware of any routing change (such as a line down or a node coming up).

5. Limits the number of nodes a packet can visit. This prevents old packets from cluttering the network.

There are two types of Network Protocol messages: data messages and control messages. Data messages carry data from the Transport Level Protocol. The Network Protocol adds a packet route header to the Transport Level messages. Control messages exchange information between Network Protocol modules in adjacent nodes to maintain the routing tables.

Routing Algorithm Requirement. One of the constraints on the EWNET design was that it start with a minimal set of capabilities while retaining the flexiability to be expanded if future requirements demand such an expansion. Therefore, the routing algorithm initially implemented was relatively simple, while retaining the capability to be expanded if the need arises. The routing algorithm consist of seven processes that must be performed:

1. Decision. The decision process selects routes to each destination in the network. It consist of a connectivity algorithm that maintains path lengths, and a traffic assignment algorithm that maintains path costs. Path length is the sum of the hops along a path between two nodes. Line cost is a positive integer value associated with using a line, and path cost is the sum of the line costs along a path between two nodes. When a routing node receives a Routing Packet, the routing node executes the two decision algorithms. Executing both algorithms is required since a packet might have a choice of two paths, both having equal cost but less hops required, or having equal hops required but less cost per line. This execution results in updating the databases used to determine packet routes.

2. Update. The update process constructs and propagates Routing Packets. The update process sends Routing Packets to adjacent nodes as required by the decision process and periodically to ensure the integrity of the routing databases.

3. Forwarding. The forwarding process supplies and manages the buffers necessary to support packet route-through to all destinations.

4. Select. The select process performs a table look-up to select the output line for the packet. If a destination is unreachable, this process discards the packet.

5. Receive. The receive process inspects a packet's route header, dispatching the packet to the appropriate Network Level or Transport Level module.

6. Congestion Control. This implementation of the EWNET Network Level Protocol does not perform any congestion control functions upon the network.

7. Packet Lifetime Control. A loop detector prevents excessive packet looping by discarding packets that have visited too many nodes.

As the topology of EWNET evolves this simple adaptive routing algorithm will require updating to a more complex distributed adaptive routing algorithm (Ref. 11:235-237).

Data Link Level Protocol Requirements. This protocol level is modeled after the High Level Data Link Control (HDLC) (Ref. 12:208-232). The overview DFD for this protocol level is shown in Figure 13, page 72. Also, Table 9 shows the process hierarchy for this

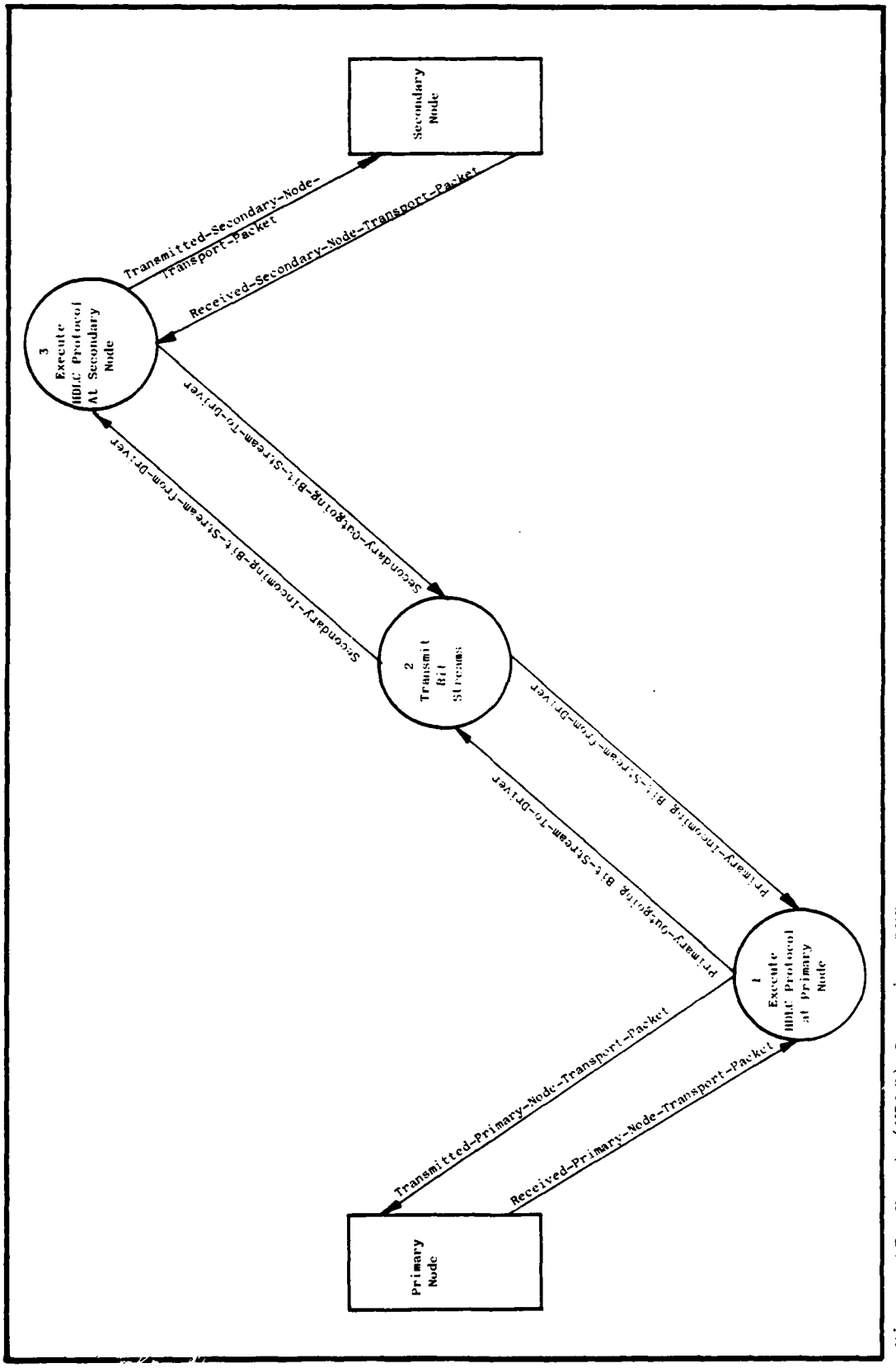


Figure 13 Ewnet (HDLC) Overview DFD

set of DFDs.

Table 9---Data Link Level Protocol Process Hierarchy

1.0 Decode HDLC Protocol at Primary Node

1.1 Decode and Sync Incoming Bit Stream

1.1.1 Check for and Remove Pad Sequence

1.1.2 Check for Startup

1.1.3 Locate Two Consecutive Sync Bytes

1.1.4 Decode Packet Header

1.1.5 Check for QSYNC

1.2 Frame Secondary Incoming Packets

1.2.1 Check Header Length

1.2.2 Check BLKCK1

1.2.3 Check Data Length (Count)

1.2.4 Check BLKCK2

1.2.5 Check ENQ Header Length

1.2.6 Check BLKCK3

1.2.7 Generate CRC Remainder

1.2.8 Check Select Bit

1.3 Execute Incoming Control Packet

1.3.1 Decode ENQ Packet

1.3.2 Set NAK Transmit Flag

1.3.3 Set Negative Acknowledge Flag, Reset Timer

1.3.4 Check STRT Select Bit

1.3.5 Check STACK Select Bit

1.3.6 Check NUM Field to R

1.3.7 Set Acknowledge Flag, Reset Timer

1.3.8 Count Errors (NAK)

1.4 Execute Outgoing Control Packet

1.4.1 N=N+1, Reset Timer

1.4.2 Generate Negative Acknowledge Packet

1.4.3 Generate Start Packet

1.4.4 Generate Start Acknowledge Packet

1.4.5 Generate Acknowledge Packet

1.4.6 Generate REP Packet

1.4.7 Clear Negative Acknowledge Flag

1.4.8 If Initialization, then Pad Packet

1.4.9 Clear Acknowledge Flag

1.4.10 Clear SREP Flag

1.5 Execute Data Packet

1.5.1 Process NUM Field

1.5.2 Process RESP Field

1.6 Execute Maintenance Packet

1.6.1 Generate Maintenance Mode Packets

1.6.2 Generate Maintenance Packet

1.7 Frame Primary Information Packets

1.7.1 Add 4 Sync Bytes to Packet if QSYNC

1.7.2 Count Data Field ADD BLKCK2 / Count

1.7.3 Add Remaining Header Fields

1.7.4 Set OSYNC Bit

1.7.5 Count Header Field and Add BLKCK1

1.7.6 Piggyback Acknowledge Received: Set RESP

1.7.7 Check Modulo 256

1.7.8 Increment Packet Count

1.7.9 Check Retransmit

1.7.10 Add 4 Sync Bytes to Data Packet if QSYNC / INIT.

1.7.11 Load and Delete Memory

1.8 Start Reply Timer

1.8.1 If STRT, STACK, or Data Packet then Start Timer

1.8.2 Reset Timer

1.8.3 Count Errors (Time)

1.8.4 Identify Timeout Packet

2.0 Transmit Bit Streams

3.0 Execute HDLC Protocol at Secondary Node

This layer of the EWNET Protocol Structure controls the operation of the physical link, maintaining the integrity and sequentiality of data transmitted over a communication channel. This Data Link Protocol is a byte-oriented link control procedure that operates in full-duplex synchronous mode and supports point-to-point communications. This protocol is concerned with the logical transmission of data grouped into physical blocks known as data packets. The primary function of the

protocol being the exchange of these data packets while ensuring their correctness. Each data packet is assigned a number to ensure proper packet sequencing at the receiver. The numbering begins with number one after initialization and is incremented by one (modulo 256) for each subsequent data packet. The receiver acknowledges the correct receipt of data packets by returning the packet number as a response. Acknowledgement of data packet n implies acknowledgement of all data packets sent up to and including data packet n. Additionally, the Data Link Protocol provides an envelop in which maintenance messages can be sent for down-line loading, up-line dumping, loopback testing and control of unattended remote systems. The protocol uses retransmission to recover from errors. The error recovery mechanism uses timeouts and control packets to resynchronize and trigger retransmission. Figure 14, page 77, is the Data Link Level Protocol high-level DFD for EWNET, low-level DFDs are supplied in Appendix C.

The Data Link Protocol uses three types of packets: data packets, control packets, and maintenance packets. All data is sent over the physical link in numbered data packets. Responses and control information are returned within unnumbered control packets. In the maintenance mode, the protocol provides a maintenance packet envelop for boot strapping, dumping and link testing. The initial field of each packet is a special ASCII character (SOH, ENQ, or DLE respectively).

The Data Link Level Protocol consist of three functional components: Framing, Link Management, and Packet Exchange.

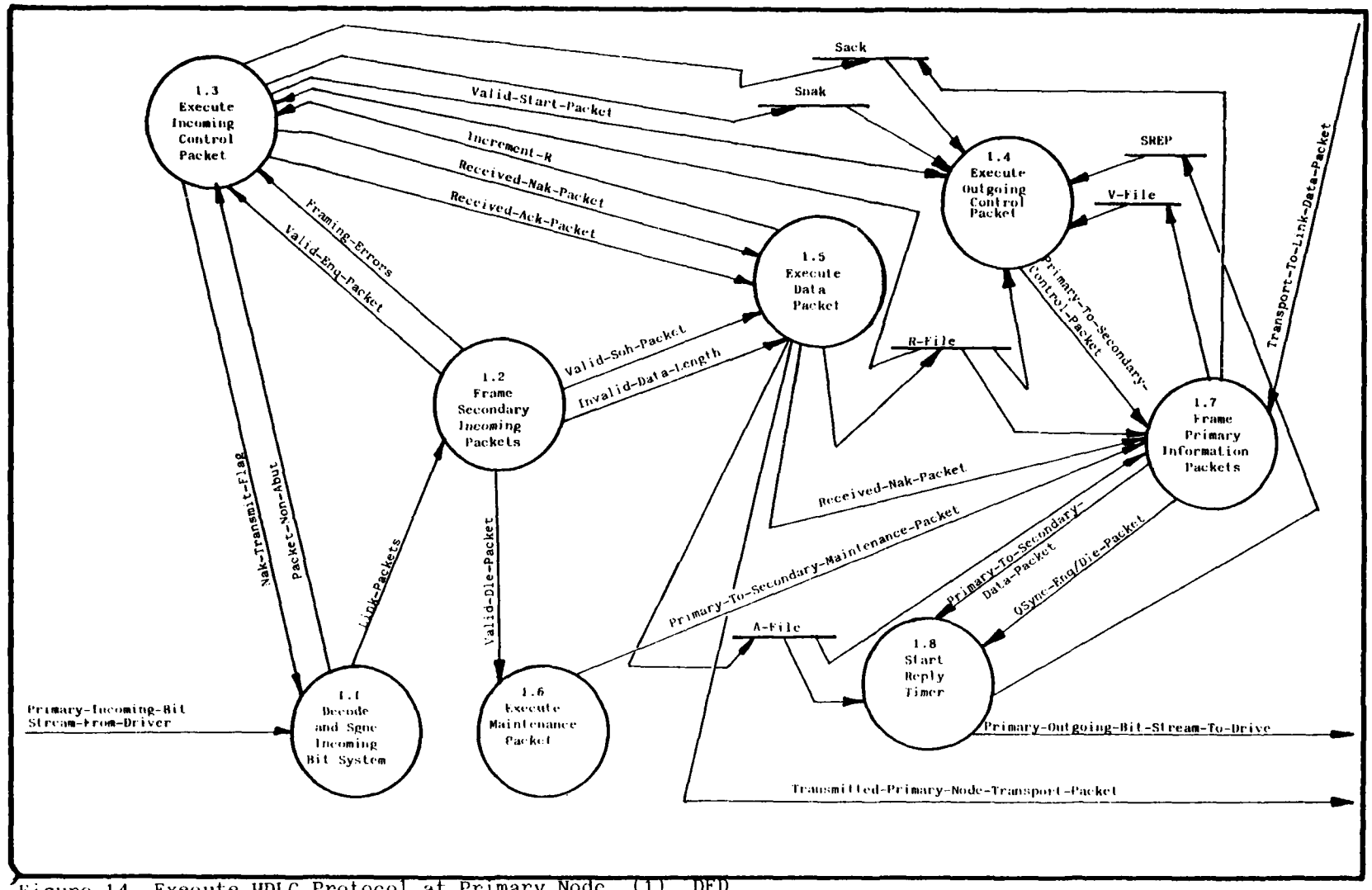


Figure 14 Execute HDLC Protocol at Primary Node (1) DFD

Framing is the process of locating the beginning and end of a packet at the receiving end of a link (Ref. 12:201-202). It requires synchronization: locating a certain bit, byte or packet and then operating at the same rate as the bit, byte, or packet. Framing is achieved when data is synchronized at the bit, byte, or packet levels (Ref. 12:330). Bit synchronization is done by the modems and interfaces on the link. Byte synchronization involves locating the proper 8-bit window in the bit stream. This is accomplished by a sync character search. Packet synchronization is achieved by searching for one of the three special starting bytes (SOH, ENQ, or DLE). After achieving byte synchronization the packet synchronization is maintained by counting out the fixed length headers, and, when required, the variable length data, based on the count field supplied in the header.

The Link Management component controls the transmission and reception on links connected to two or more transmitters and/or receivers in a given direction (Ref. 12:122-129).

The Packet Exchange component transfers the data correctly and in sequence over the link. Once framing is accomplished, this component operates at the packet level, exchanging data and control packets.

The Data Link Protocol is a positive acknowledgement retransmission protocol. For each data packet correctly received and passed to the next level, a positive acknowledgement is returned on the link, notifying the transmitter of the correct receipt of the data packet. If a data packet is incorrectly received, the data is not passed to the

user and the packet is not acknowledged. Eventually, the packet will be retransmitted. For efficiency, the Data Link Protocol may transmit several packets (up to 255) before requiring the acknowledgement of the first one. This is called pipelining (Ref. 11:153-157). If the CRC block check reveals a transmission error, the receiver returns a negative acknowledgement.

The Data Link Protocol provides three types of counters for recording and evaluating errors: Threshold counters, Cumulative counters, and Background counters. Threshold counters are used to detect persistent error conditions; Cumulative counters record overall error statistics; and Background counters provide a base on which to evaluate performance. To determine a persistent error, the protocol counts consecutive errors, notifying the user when a predefined threshold value is reached.

Physical Level Protocol Requirements. The DECNET does not require that any one physical level protocol be used (Ref. 13:300). Thus, there were, only two requirements for the physical level protocol. It had to be a widely recognized standard to minimize the difficulty in adding new hosts to EWNET and it had to be compatible with the transmission medium selected and the distances of transmission required. Thus, if these requirements could be met then the Transmit Bit Streams (2.0) process in the Data Link Level Protocol (Figure 13, page 72) could be performed.

Summary

This chapter has refined the hardware specifications and addressed the software specifications for EWNET in detail. The software specification was lengthy but, because of the partitioning and levels of abstraction made possible through the use of Structured Analysis, if was easier to comprehend than would otherwise be possible. The data dictionary contains the explicit information on these specifications and is included in Appendix C along with the high and low-level data flow diagrams. These requirements formed the basic foundation for the overall structural design approach taken in Chapter IV to design the EWNET.

IV. Design of EWNET

Introduction

The previous chapters determined the user functional requirements and translated them into detailed hardware and structured software specifications. In this chapter, the hardware requirements specification was employed along with supporting background information to develop the hardware design. The software for EWNET was also designed by selecting the specification of a commerically available network design (DECNET). As a part of the software and hardware design, Appendix D contains a complete EWNET Network Profile for use by the Electronic Warfare Engineering Branch as a means of obtaining the correct commerically available network configuration to meet their present and future needs. This chapter also presents diagrams that depict the exact interaction of the Host-to-Node network configuation selected for the EWNET design.

Hardware Design

Topology. The topology shown in Figure 15, page 82, was chosen for EWNET for two reasons. First, the basic loop architecture of the nodes allows the initial routing algorithm to be "simple" since a message has only two possible paths to its destination node. This gives the Electronic Warfare Branch Laboratory the minimum operational

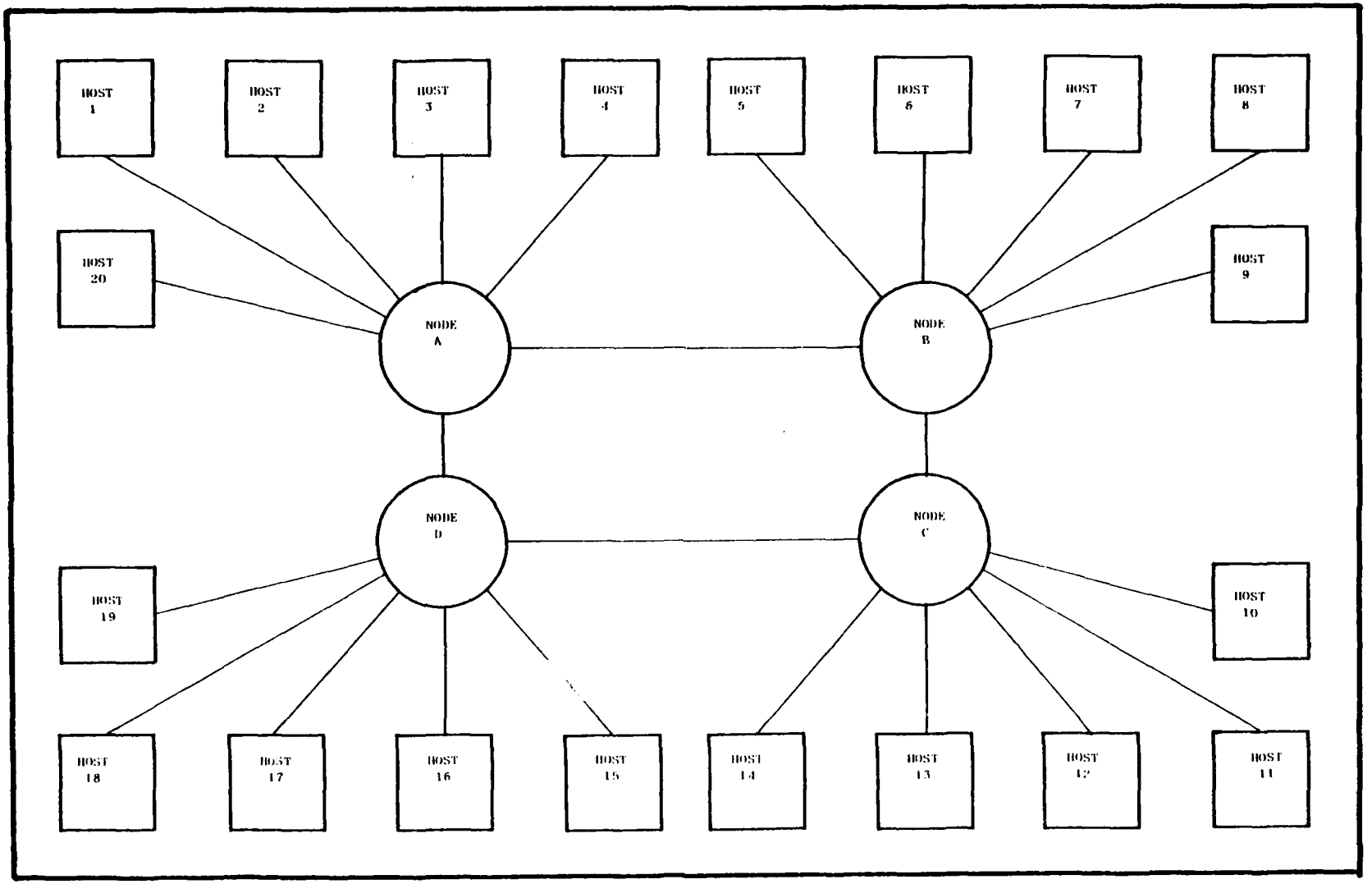
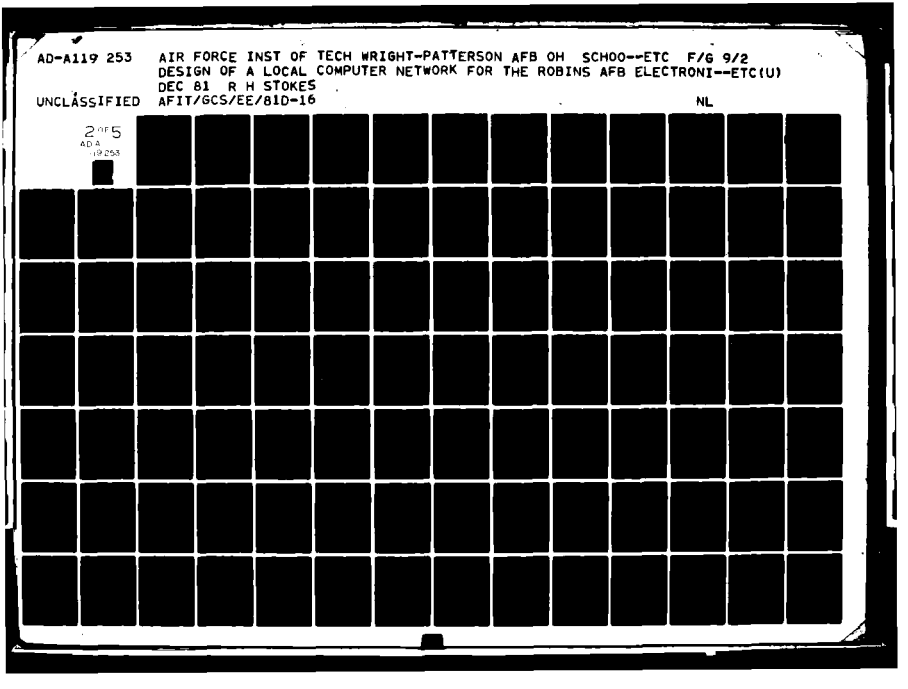


Figure 15 Basic Ewnet Topology

capability, as identified in the functional requirements, in the least amount of development time; thereby, allowing additional time to evaluate and develop a more sophisticated routing algorithm to handle more complex future topologies. Second, to maintain a cost-effective initial network, the loop network has only one more link between nodes than a minimum spanning tree, since deleting one of the links, results in a linear connection of the node computers. This linear connected network then constitutes a minimum spanning tree since additional node-link deletions would result in a non-connected network. As the topology of EWNET expands to include an additional node computer, it will no longer be possible to take advantage of the cost-effectiveness derived from the minimum spanning tree concept, since the basic loop architecture will be replaced by a topology which will give maximum reliability and response time to the network.

The star topology connecting the hosts to the node was chosen for several reasons. First, it minimizes the loop topology disadvantages of increased response time and reduced reliability due to multiple node computers connected in a single loop. Second, the topology given in Figure 15, page 82, takes full advantage of the star configuration by grouping around the same node those host that will interact the most with each other. In this configuration, much of the traffic need only pass through its entry node to the network. This decreases the average response time on EWNET and decreases the amount of traffic transmitted between nodes. However, reliability from the host's perspective is decreased since the failure of a node can cause several hosts to lose


AD-A119 253
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONI--ETC(U)
DEC 81 R H STOKES
UNCLASSIFIED AFIT/GCS/EE/81D-16
NL
2 OF 5
AD A
119253

access to the network. Also, there must be a greater transmission rate between nodes than that between the hosts and the nodes. This is to keep the nodes from acting as bottlenecks and thereby decreasing throughput to an unacceptable level. Finally, the star configuration allows any host to be able to drop out of the network without adversely affecting the performance of the network. This meets the requirement, that all ISSes must be able to operate completely independent from the network without affecting the network.

Hosts. Fourteen hosts were chosen for the initial network configuration. This number represents those EW systems within the Electronic Warfare Engineering Branch Laboratory which will require the greatest access to the network and which contain the most sophisticated and powerful minicomputers. The following list of Hosts represents a generous cross-section of the type systems which will require access to the network (present and future).

1. F-15 Tactical Electronic Warfare System (TEWS) = Integrated ISS System

2. EF-111A = Integrated ISS System

3. APR-38 = Fire control ISS System

4. Area Reprogrammable Capability (ARC) = EW system emulation ISS System

5. ALQ-131 = Jammer ISS System

6. ALQ-155 = Jammer ISS System

7. ALQ-119 = Jammer ISS System

8. ALR-69 = Radar Warning Receiver (RWR) ISS System

9. EWOLS/ECSAS = Simulation and analysis systems respectively

10. B-52 = Tail Warning ISS System

11. ALR-62 = RWR ISS System

12. ALR-46 = RWR ISS System

13. Flight Line Test Sets (FLTS) = FLTS ISS System

14. ALQ-125 = Electronic Intelligence Gathering (ELINT) ISS System

Nodes. Since there existed no requirement to be able to internally modify the network protocols once the network was established, and due to the standardization efforts of the Electronic Warfare Engineering Branch (Ref. 3), it was decided that the node computers used within the network should be compatible with the standardized ISS computer and at the same time be able to host the commercially selected software protocol design. This resulted in the selection process for the node computers being narrowed down to the Digital Equipment Corporation's PDP-11 family of computers. There still remained the management requested requirement that, if possible, the organizational Host Processor (UNIVAC-1108) functions and the Network Routing Node functions be performed by the same computer(s). Therefore, it was decided that the Network Node computers could easily perform the existing organizational Host Processor functions if the node computers were of sufficient sophistication.

The Digital Equipment Corporation's PDP-11/70 computer was selected for the EWNET Node computers. It is of sufficient sophistication to support all present and future organizational Host Processor functions (Ref. 14:261-305), it meets all Electronic Warfare Engineering Branch

Laboratory standardization efforts (Ref. 3), and it supports the Phase III routing DECNET software (Ref. 15:67). As shown in Figure 16, page 87, there will initially be three PDP-11/70 node computers. The division of functions for these computers other than network functions is given below:

1. First Floor Node = Host all off-line ISS backup assemblers.

2. Second Floor Node = Host all common ISS software tools and common databases such as the organizational threat database.

3. Third Floor Node = Host all off-line ISS (Flight Test) data reduction programs.

Transmission Medium. Due to the security and EMI requirements for EWNET, as discussed in Chapter 3, all transmission links within the network will use fiber optics. The optimum transmission rate for the network, considering the throughput and response time requirements, was 1 mbps.

Appendix D contains further details concerning the hardware design and Figure 16, page 87, shows the total hardware design.

Software Design

Introduction. The design of the software was accomplished through the specification of a commercially available network design. The Electronic Warfare Engineering Branch Management, acting in behalf of a

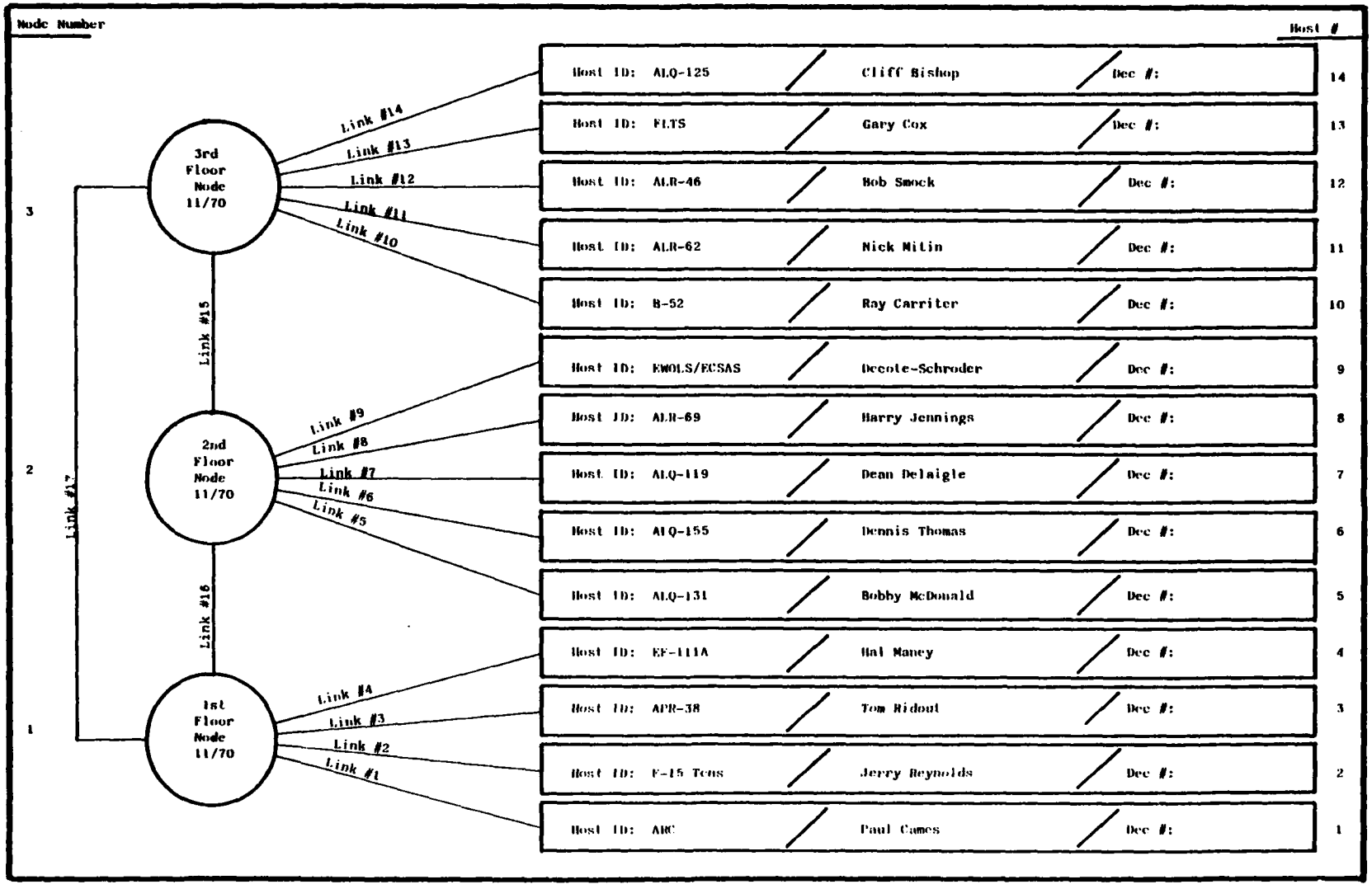


Figure 16 Initial Ewnet Configuration

then non-existing network management group, specified that the EWNET design must be obtained from an existing, contract supportable, commercially available network design that would meet all internal standardization efforts and be as compatible as possible with existing ISS standard computers. Considering these requirements the Digital Equipment Corporation's DECNET product was selected (Ref. 15:331-349).

DECNET. DECNET is the collective name for the set of software products that extend various Digital Equipment Corporation operating systems by enabling the user to interconnect these systems with each other to form computer networks. DECNET Phase II products allow point-to-point physical and logical connections. DECNET Phase III products extend the logical connectivity through routing (Ref. 16:17). Phase II and III products can be intermixed in the network.

In order to satisfy widely varying applications, DECNET allows the user to build networks from a wide range of systems and communications components. DECNET allows the user to interconnect systems using serial synchronous facilities. When configuring DECNET systems, both ends of any given link must use the same type of communications discipline running at the same line speed (Ref. 15:331-339).

DECNET is implemented from a set of layered network protocols (ISO OSI Model), each of which is designed to fulfill specific functions within the network. Collectively, these protocols are known as the Digital Network Architecture (DNA)(Ref. 16:Vol 1).

DNA modules in a layer typically use the services of modules in the layer immediately below. Some layers may contain more than one type of module (as is the case in Phase II DECNET Protocols). Each module specified by DNA operates as a "black box". That is, the operation within the black box is transparent to other layers and to equivalent modules in other nodes. The architecture does not define specific code for implementing modules. It only defines specific operations that the modules must perform. The architecture specifies two kinds of relationships between modules:

1. Interfaces. Interfaces are the relationships between different modules that are usually in the same node.

2. Protocols. Protocols are the relationships between equivalent modules that are usually in different nodes.

The architecture specifies common error reporting, operational parameters, and counters that certain layers must maintain. This standardization ensures that maintenance, error logging, and network management can take place consistently. Finally, DNA is an open-ended architecture. Future phases of DNA may model additional layers, additional modules within existing layers, or alternative models for certain layers. A brief description of DECNET Phase II and Phase III architectures follows.

DECNET Phase II. Figure 17, Page 90, shows the building of information for each layer of the DECNET Phase II DNA. Each protocol

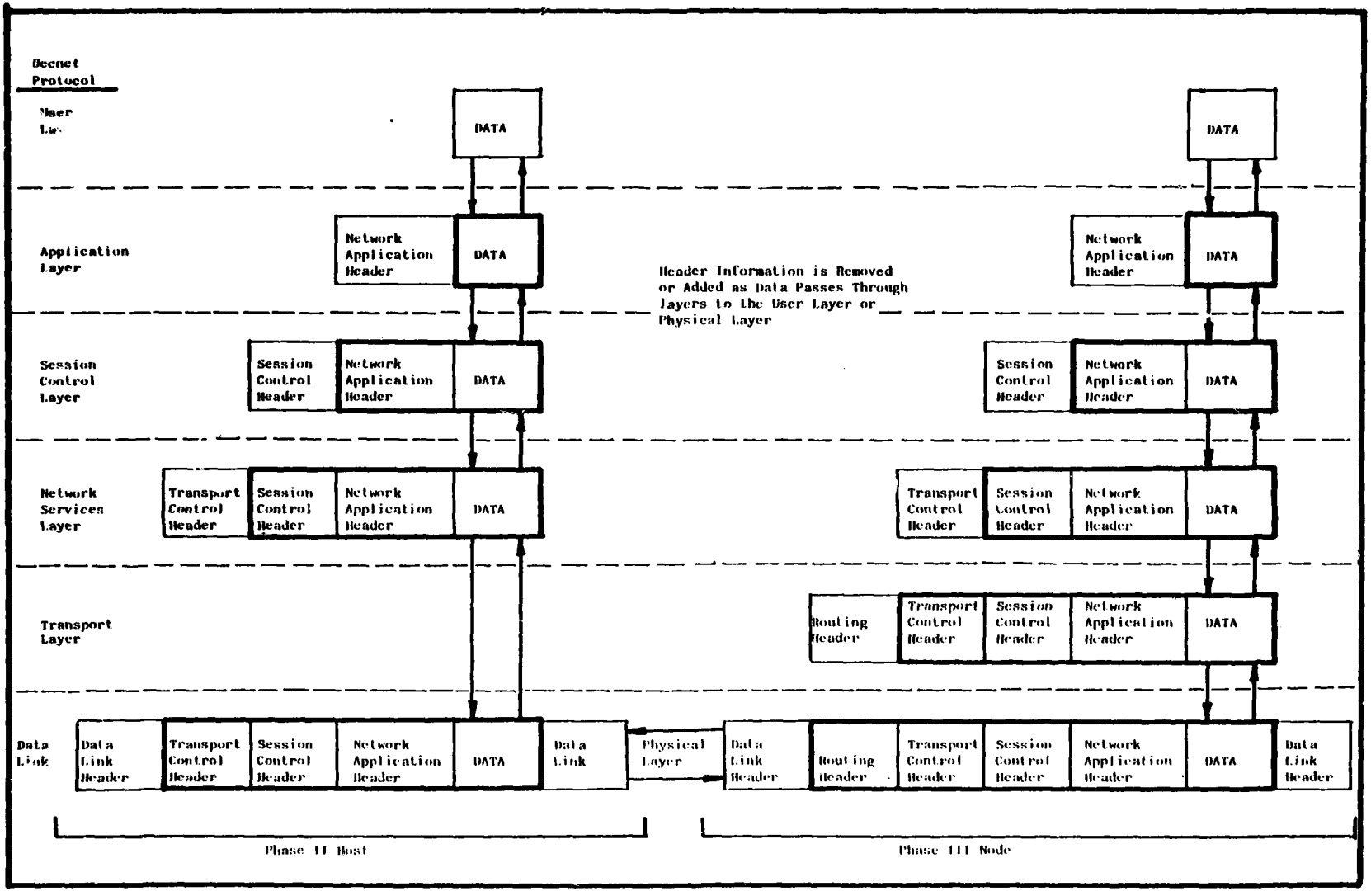


Figure 17 Ewnet Protocol Building

layer present is defined as follows (Ref. 16):

1. User Layer. This layer contains all user-supplied functions. It is the highest layer in the Phase II DNA structure (Ref. 16:Vol I).

2. Network Application Layer. This layer provides the network functions for the user layer. Modules in this layer include network remote file access modules, a remote file transfer utility, and a remote system loader module. The protocol used for remote file access and file transfer is the Data Access Protocol (DAP)(Ref. 16:Vol II).

3. Network Services Layer. This layer provides a location-independent communication mechanism for both the user layer and the network application layer. Two network application modules may communicate with each other by means of the network services layer regardless of their network locations. The protocol used between network service modules is the Network Services Protocol (NSP)(Ref. 16:Vol III).

4. Transport Layer. This layer provides a mechanism for the network services layer to send a unit of data (a packet) from any node in a network to any other node in the network. This layer, while conceptually seperate from the network service layer, is sufficiently simple in Phase II DNA that its specification is encompassed by the NSP specification described above (Ref. 16:Vol III).

5. Data Link Layer. This layer provides the transport layer with an error-free communication mechanism between adjacent nodes. The data link module specified for this layer implements the Digital Data Communications Message Protocol (DDCMP). The functions provided by this layer are independent of communication facility characteristics (Ref. 16:Vol IV).

6. Physical Link Layer. Modules in the physical link layer manage the physical transmission of data over a channel. The functions of modules in this layer include monitoring channel signals, clocking on the channel, handling interrupts from the hardware, and informing the data link layer when a transmission is complete. Implementations of this layer encompass parts of device drivers for each communications device as well as the communications hardware itself. The hardware includes devices, modems, and lines. In this layer, industry standard electrical signal specifications such EIA RS-232-C or CCITT V.24 operate rather than protocols (Ref. 16:Vol V).

The services between modules in the physical link and data link layers are less sophisticated than those provided by the transport layer modules. The modules that reside in the physical link and data link layer provide services for moving data from a given node to an adjacent node only. The modules in the transport and higher layers provide services for moving data from a given node to any other node in the network.

DECNET Phase III. Figure 17, page 90, shows the building of information as data passes through the Phase III DNA layers. Each of the Phase III DNA layers are defined below:

1. User Layer. The user layer contains most user-supplied functions. It also contains the Network Control Program, a network management module that gives system managers access to lower layers to control and observe the network from a terminal (Ref. 17:Vol I).

2. Network Management Layer. Modules in the network management layer provide user control of and access to operational parameters and counters in lower layers. Network management also performs down-link loading, up-line dumping, and test functions. In addition, the network management performs event logging functions. This layer is the only one that has direct access to each lower layer. The protocols for this layer are (Ref. 17:Vol II):

A. Network Information and Control Exchange (NICE) Protocol. This is used for triggering, down-line loading, up-line dumping, testing, reading parameters and counters, setting parameters, and zeroing counters.

B. Event Logger Protocol. This is used for recording significant occurances in lower layers. An "event" could result from a line coming up, a counter reaching a threshold, a node becoming unreachable, and so on.

3. Network Application Layer. The network application layer provides generic services to the user layer. Services include remote file access, remote file transfer, and resource managing programs. This layer contains both user- and Digital-supplied modules. The following modules execute simultaneously and independently in this layer (Ref. 17:Vol III):

A. Data Access Protocol (DAP). This is used for remote file access and transfer.

B. Loopback Mirror Protocol. This is used for network management logical link loopback tests.

4. Session Control Layer. The session control layer defines the system-dependent aspects of logical link communication. Session control functions include name to address translation, process addressing, and process activation and access control. The module at this level is called the Session Control Protocol (Ref. 17:Vol IV).

5. Network Services Layer. The network services layer is responsible for the system-independent aspects of creating and managing logical links for network users. The Network Services Protocol (NSP) module performs data flow control, end-to-end error control, and the segmentation and reassembly of user messages (Ref. 17:Vol V).

6. Transport Layer. Modules in the transport layer route user data, contained in packets, to its destination. The Transport Protocol module also provides congestion control and packet lifetime control (Ref. 17:Vol VI).

7. Data Link Layer. This is DDCMP as described previously for Phase II DNA (Ref. 17:Vol VII).

8. Physical Link Layer. This module is the same as described previously for Phase II DNA (Ref. 17:Vol VIII).

The three higher layers each interface directly with Session Control for logical link services. Each layer interfaces with the layer directly below to use its services. The User Layer interfaces directly with the Network Application Layer as well. In addition, Network Management modules interface with each lower layer directly for access and control purposes. Finally, Network Management interfaces directly with the Data Link Layer for service functions that do not require logical links.

EWNET Design. The primary purpose of a network is to pass data from a source in one node to a destination is another. Because EWNET is designed around the Phase II and III DNA structure, it is important to understand how data flows through the DNA layers and between EWNET nodes. Data traveling from one node in a network to another passes from a source process in the user layer down through each layer of the appropriate-phase DNA hierarchy of the source node or host before being

transmitted across a line. If the destination node or host is not adjacent, the data must then travel up to the Transport Layer of the adjacent node, where it is routed (or switched), sent back down through the two lower layers, and transmitted across the next line in the path. The data keeps traveling in this manner until it reaches its destination node or host. At this node or host, the data passes up the hierarchy of layers to the destination process.

All hosts attached to EWNET will contain Phase II DNA software, while all EWNET nodes will contain Phase III DNA software. This configuration is appropriate for EWNET since the use of Phase II DNA software in the hosts will reduce the amount of network software that must be present (homogeneous network) in the host. This satisfies the requirements that the hosts (ISS computers) contain the least amount of network software possible; and that host should not be allowed to perform routing functions. Since the EWNET nodes are the center of the star topology, they will be required to perform network packet routing and thus will require the Phase III DNA software. Figures 18, 19, and 20 on pages 97, 98, and 99, are the data flow graphs for EWNET.

Summary

This chapter has translated the hardware and software requirements into a commercially available system design. The hardware design specifies the topology to be used, the hosts to be included, the

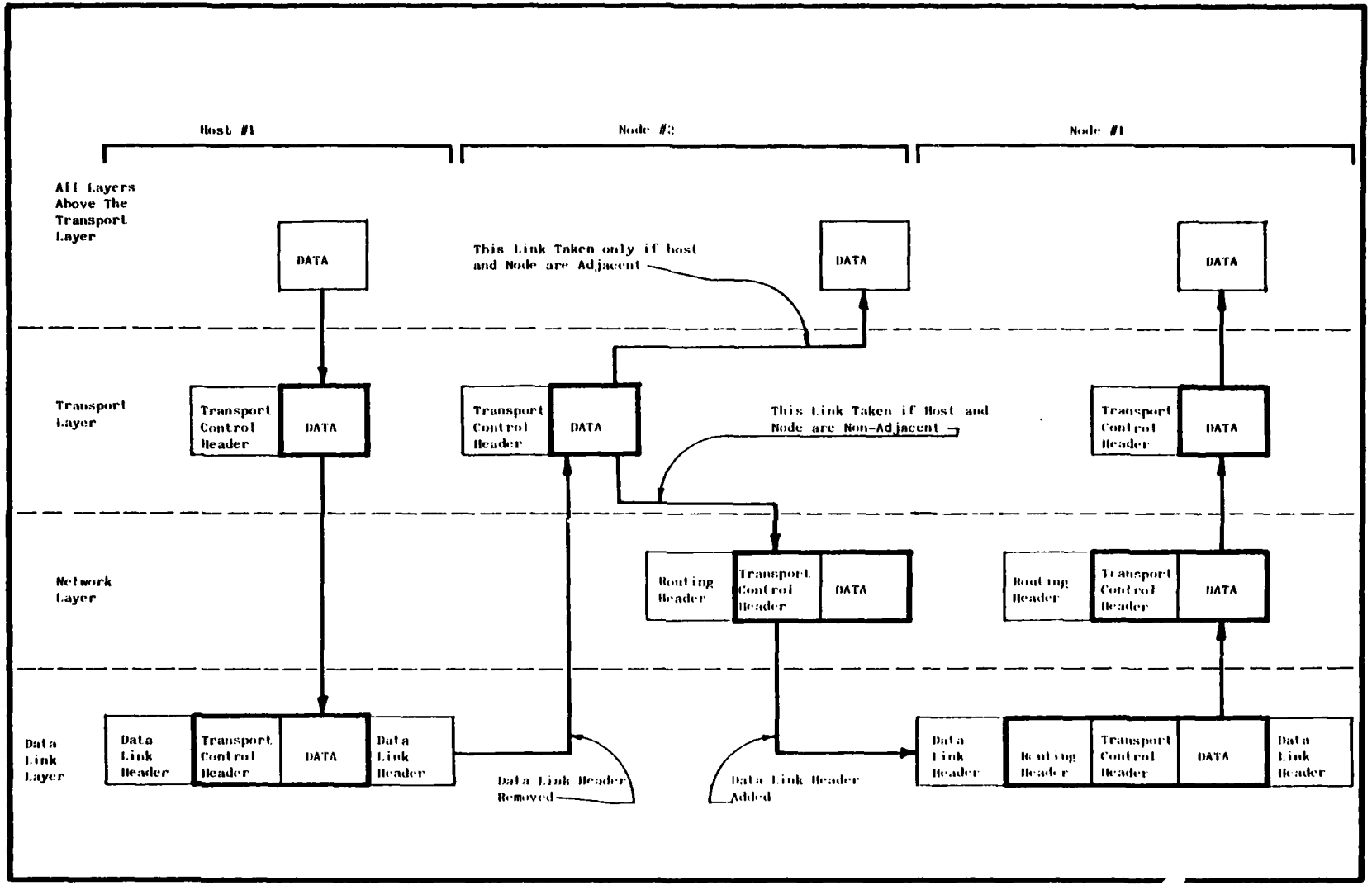


Figure 18 Host-to-Host/Node-to-Host Data Flow

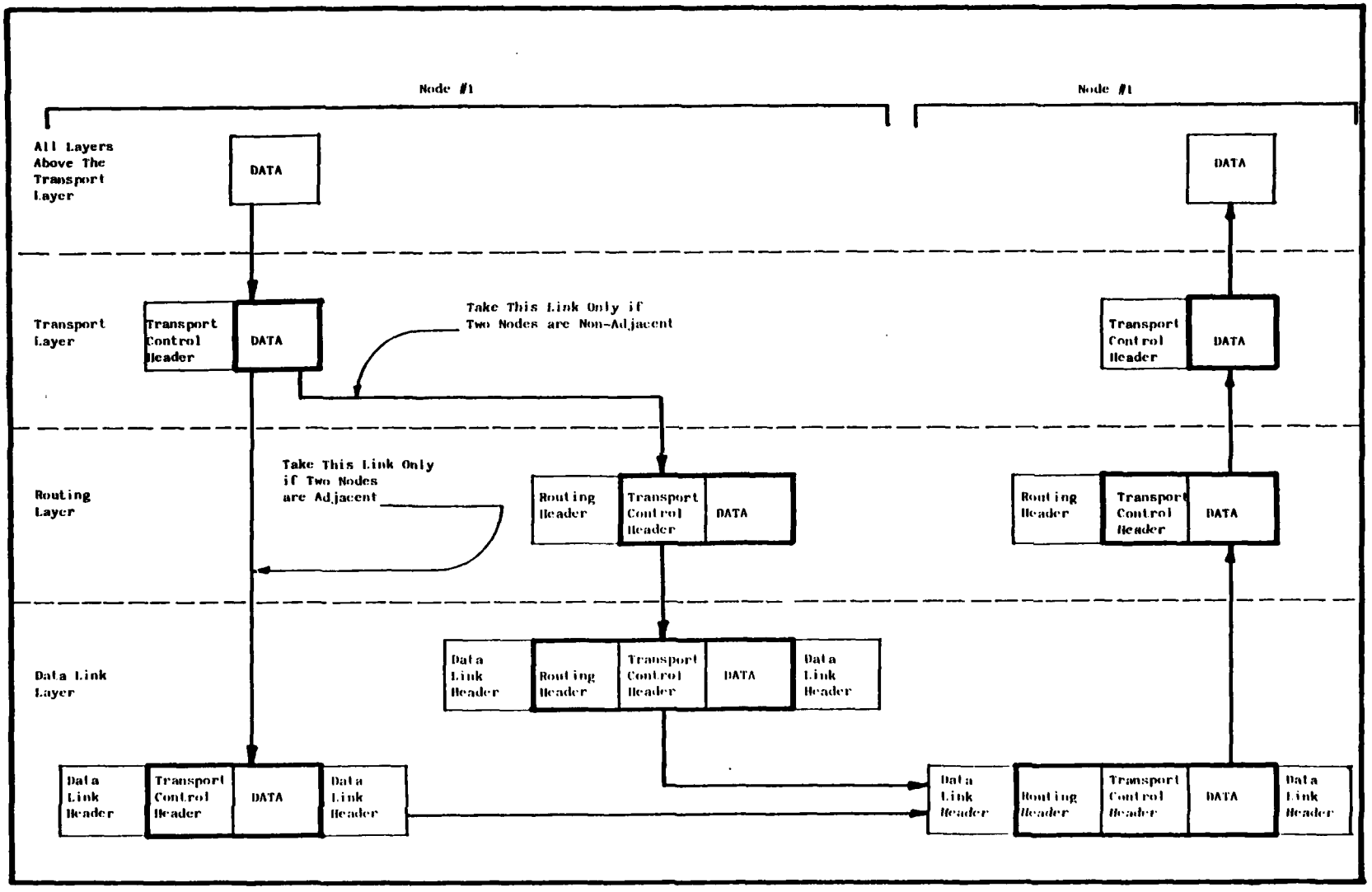


Figure 19 Node-to-Node Data Flow

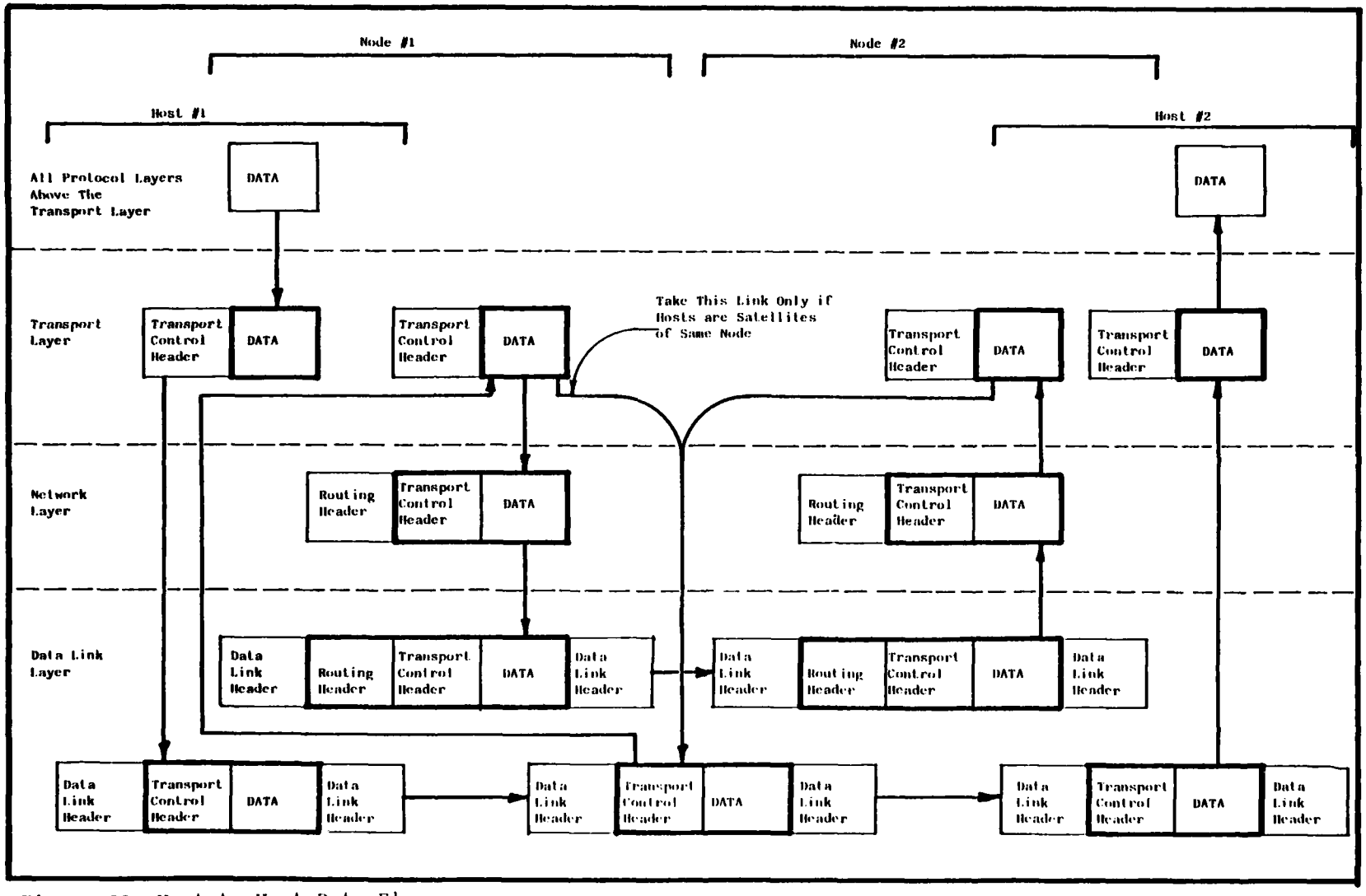


Figure 20 Host-to-Host Data Flow

specific node to use, and the transmission medium to be used for each EWNET link. Each of these design decisions is based upon the requirements analysis and supported by background data gained from researching the various design options. The software design is based upon the structured specification in Appendix C and was derived primarily by studing the commercially available network products such as IBM's SNA, Xerox's ETHERNET, and Digital's DECNET. Once the network product best suited to EWNET functional requirements was chosen, further refinement of the commericallly available network product was required to fine tune the network configuration to the EWNET requirements. The Digital Equipment Corporation's DECNET was chosen and refined because it best suited EWNET's functional requirements and because the DECNET was designed around the X.25 communications protocol (Ref. 15:348). Additionally, DECNET is completely supported by the Digital Equipment Corporation's Internets product (Ref. 15:340-349). The Internet products are data transfer facilitators (not hardware emulators) that provide mechanisms for interchanging data with IBM's SNA protocol structure and remote batch workstation, UNIVAC's remote UN1004/NTR batch, and CDC's interactive MUX200 batch. Finally, Digital Equipment Corporation is in the process of implementing Packetnets (public packet switching networks) based on the X.25 protocol throughout Canada, the United States, and France; and initial local computer DECNET networks are being designed by Digital Equipment Corporation that will be based entirely upon the ETHERNET concept.

Due to the above discussion it is apparent that Digital Equipment

Corporation's DECNET is not just another network design; is is a complete networking concept which allows future growth as technological advances are made.

The complete structured design of EWNET is contained in the Network Profile located in Appendix D (Refs. 18;19). Appendix D was included as a tool to be used by the EWNET Management group in the procurement of the EWNET. Appendix D was written in the exact format required by the Digital Equipment Corporation, but does not contain all data necessary to procure the network at this time since this thesis study was concluded before complete information on all hosts became available (had not been procurred).

This chapter allocated all off-line host software modules to particular node processors in the network.

V. Conclusions and Recommendations

Conclusions

This investigation, the development of EWNET, was based upon the actual requirements of the Electronic Wartare Engineering Branch Laboratory. The user interviews were used both to determine what the exact user requirements were and to provide a vehicle to document these requirements. From these user identified requirements the functional requirement specifications of EWNET were determined.

The EWNET functional requirement specifications were broken down into the hardware specification and the software structured specification. This portion of the investigation was the most time-consuming. The process of determining these specifications required that iterative design decisions be made as the specifications evolved.

The time spent developing the functional requirement specifications proved to be worthwhile once the design phase was started. Because of the amount of partitioning that had to be done to arrive at the specification level, the design proved to be straightforward. The evaluation and selection of a commercially available network product and the tuning of this network configuration lead to a network design that was unique with respect to the EWNET functional requirements.

The network was designed in such a way to minimize the amount of

protocol software residing in each ISS so that minimum effect would be felt by each ISS at implementation time. The Engineering Branch Host Computer functions and the EWNET functions were combined to eliminate the need for virtual terminal protocols. The most difficult portion of the investigation concerned the vast amount of information existing on network designs, forcing extensive reading on the part of the author.

The implementation and testing of the EWNET configuration is in process at the Electronic Warfare Division, Robins AFB, Georgia. The implementation is proceeding in steps; each star configuration is being set up and tested before the core loop configuration will be added. This approach is giving the ISS lead engineers time to familiarize themselves with the benefits of using the network.

Recent news from Robins AFB indicates that the EWNET is already proving to be a valuable tool to the Electronic Warfare Division; the time spent in obtaining databases from other similar ISSes has been significantly reduced, thereby freeing up valuable time for engineers to perform other task.

In summary, through the user interviews, Structured Analysis techniques, and Structured Design techniques, a top-down design of EWNET was achieved. All primary requirements of the Electronic Warfare Engineering Branch Laboratory for file transfer capability, resource sharing and ISS independence have been met in the design. The structured approach allowed the interfaces between the layers of protocol to be clearly defined and together with the functional

requirement specifications allowed the selection of an appropriate commercially available network product to be signigicantly simplified, leaving only network enhancements to be considered in follow-on investigations.

Recommendations

Because the selection of a design for EWNET was limited to a commercially available network design which is already in the implementation and testing phases, follow-on investigations will be limited to enhancements to the EWNET. The exact enhancements and specific task involved which the Engineering Branch Laboratory would be interested in are listed below:

1. Data link to Eglin AFB, FL. Math Laboratory.

   a. Determine best approach (satellite or landline).

   b. Develop and implement the necessary virtual terminal protocol.

2. Standardize all outputs from the network into one format.

3. Automate all ISS and EWNET documentation.

4. Implement a network command terminal capability so that standalone terminals can be added to the network.

5. Evaluate DECnet statistical network data and make recommendations for additional capability (i.e., software monitors, hardware monitors, or standalone statistics collecting node).

6. Evaluate the EWNET routing algorithm when the network grows to include an additional node.

a. Design new algorithm

b. Test algorthim using a simulation of the EWNET

c. Implement new algorithm

A simulation for an AFLC Bulk Data Network has been formulated in another research project at the Air Force Institute of Technology (Ref. 20). This simulation was written by a fellow Robins, AFB employee using Pascal to represent a specified network design. But, in conversations with the simulation author, it was determined that slight modifications (packet length, etc.) could result in the tailoring of the simulation to represent the EWNET cofiguration. This simulation could then be hosted on any of the computers within the Engineering Branch Laboratory and act as a valuable testing tool in the evaluation of future enhancements to the EWNET. Finally, it is suggested that extensive study and evaluation of the DECnet-ETHERNET concept be undertaken once the Digital Equipment Corporation makes the documentation available. The study should focus on the advantages to the Engineering Branch Laboratory of implementing this protocol structure on EWNET.

Bibliography

1. Hobart, William C. Design of a Local Computer Network for the Air Force Institute of Technology Digital Engineering Laboratory. MS Thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, March 1981.

2. Vogler, F. H. and J. A. Copland. Standardized Integration Support Station System Study. Georgia Institute of Technology Study. Engineering Experiment Station, Altanta, Georgia, May 1980.

3. Georgia Institute of Technology, Engineering Experiment Station, Systems Engineering Laboratory. Standardized Integration Support Station System Specification. Altanta, Georgia, 1980.

4. McQuillan, John M. "Local Network Architecture", Computer Design, 4:18 - 26 (May 1981).

5. Dineson, Mark A. "Broadband Local Networks Enhance Communication Design", Electronic Design News, 12: 77 - 85 (March 1981).

6. Martin, James. Design of Man-Computer Dialogues. Englewood Cliffs, N. J. : Prentice-Hall, Inc., 1973.

7. Air Force Logistic Command Headquarters Staff Letter. Wright-Patterson AFB, Ohio. 1 April 1981.

8. Levy, Walter A. "Too Many Networks, Not Enough Gateways", Mini-Micro Systems. 9 : 104 - 109, (September 1980).

9. Weinbery, Victor. Structured Analysis. New York, N. Y. : Yourdon, Inc., 1979.

10. TRW Defense and Space Systems Group. Long Range Logistics Support Plan for Embedded Computer Systems---Phase III Dratt. Redondo Beach, California, April 1981.

11. Tanenbaum, Andrews. Computer Networks. Englewood Cliffs, New

Yersey: Prentice-Hall, Inc., 1981.

12. Davies, D. W., D. L. A. Barber, W. L. Price, and C. M. Solomonides. Computer Networks and Their Protocols. New York, N. Y.: John Wiley and Sons Ltd., 1979.

13. Digital Equipment Corporation. VAX 11 Software Handbook. Description of VAX-11/780 Software. Digital's Sales Support Literature Group, 1980.

14. Digital Equipment Corporation. PDP 11 Processor Handbook. Description of PDP-11/04/24/34A/44/70 Processors. Digital's Sales Support Literature Group, 1981.

15. Digital Equipment Corporation. PDP 11 Software Handbook. Description of PDP-11 Software Products. Digital's Sales Support Literature Group, 1980-81.

16. Digital Equipment Corporation. Digital Network Architecture II Protocol Specifications: Volumes I, II, III, IV, and V. Description of the Digital Network Architecture - Phase II Implementation of All Protocol Layers. Digital's Sales Support Literature Group, 1980.

17. Digital Equipment Corporation. Digital Network Architecture III Protocol Specifications: Volumes I, II, III, IV, V, VI, VII, and VIII. Description of the Digital Network Architecture - Phase III Implementation of all Protocol Layers. Digital's Sales Support Literature Group, 1981.

18. Digital Equipment Corporation. Peirpheral Handbook. Descriptions of all Peripheral Equipment Manufactured by the Digital Equipment Corporation. Digital's Sales Support Literature Group, 1981.

19. Digital Equipment Corporation. Terminals and Communications Handbook. Descriptions of all the Terminals and Interface Products Manufactured by the Digital Equipment Corporation. Digital's Sales Support Literature Group, 1980.

20. Stewart, Stephen. Simulation of the AFLC Bulk Data Network. MS Thesis, Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1981.

Appendix A

User Interview Results

This appendix contains a compilation of the results of the user interviews that were conducted in the requirements analysis phase of the investigation at Robins AFB, Georgia on 10, 11, and 12 June 1981. These results provided the basis for the specification of the EWNET functional requirements in Chapter II.

Contents

To: MMRR (Mr. Joe Black)

Subject: AFIT Thesis : Electronic Warfare Local Computer Network (EWNET)

From : AFIT/ENE (Mr. Robert H.Stokes, Box 4083)

1. Attached you will find 25 copies of the interview I am using to obtain data necessary to design the Electronic Wartare Engineering Branch Local Computer Network (EWNET).

2. These copies are provided for distribution to yourself and the MMRR sections listed below:

1. MMRRA
2. MMRRC
3. MMRRI
4. MMRRV
5. MMRRW (Info Copy Only)
6. MMRRF (Info Copy Only)

I wish each section chief to retain a copy and send the extras to the following system lead engineers:

1. AN/ALR-46
2. AN/ALR-62
3. AN/ALR-69
4. AN/ALQ-119
5. AN/ALQ-125
6. AN/ALQ-131
7. AN/ALQ-155
8. AN/APR-38
9. B-52
10. F-15 TEWS

11. EF-111A

12. FLTS

13. EWOLS/ECSAS

14. UNIVAC-1108

15. Engineering Branch Standardization Group

Plus any systems the section chiefs feel should be included in this list. Feel free to make additional copies of the interview if the need arises.

3. Each system engineer should review the EWNET interview, answering those questions that are self explanatory, within the next two weeks. I will be arriving at WR-ALC on the morning of 10 June 1981, and will be available to interview each system engineer and to answer questions about the interview portions that are not self explanatory. I will only be able to spend 3 days at WR-ALC before returning to Ohio. Therefore, I hope that the two weeks before I arrive will be ample time for all users to review the interview and be prepared for my arrival. In the interest of expediency, I would appreciate it if each section chief would query the system engineers in his command and meet with the other section chiefs to setup an interview schedule for me. I think it will take approximately 50 minutes per interview if each system lead engineer is interviewed separately. If the section chief wishes, I can be prepared to interview all system lead engineers in his section at one time in a group discussion. The interview schedule should be delivered to Mr. Black's office COB Tuesday 9 June 1981, for immediate pickup by Mr. Stokes on the morning of 10 June 1981.

4. I am fully aware of the limited time and resources at your disposal and hope that what I have requested will not cause undue hardships on anyone. I only wish to design a network that will meet everyone's needs within the Engineering Branch, while providing the maximum feasable support for the ISSes. I can only obtain this goal if I have your help, without it I must rely on my own limited knowledge of all the system requirements. Your efforts in this area are much appreciated.

Robert H. Stokes
MSEE Computer Systems

EWNET INTERVIEW

Introductory Narrative

I am in the Preliminary design phase of the Electronic Warfare Branch Local Computer Network (EWNET) and am attempting to determine the uses envisioned for EWNET as well as all functional requirements associated with it. Therefore, I would appreciate any help that you can give me in determining these requirements. Hopefully, the questions that I have prepared in the interview outline will provide a framework within which you can communicate your ideas to me in this area.

The interview is divided into four sections. The first section lists typical uses of local computer networks, asks you to evaluate the benefits of having the capability for each use on EWNET, and then asks you to specify which uses you would like to see implemented first. The second section lists some of the functional requirements that must be determined and asks specific questions dealing with each of these requirements. At the end of this section, you will be asked to rate each of the functional requirements on a scale from "DOES-NOT-APPLY" to "MUST HAVE". The third section requests that you express any ideas that you have concerning EWNET that were not expressed by your responses to the questions in the first two sections. Finally, the fourth section contains questions related to design-oriented requirements and deals with network management. Users are not required to answer these questions, only section and branch management need respond, although all

inputs will be appreciated.

Name of the Person Interviewed_____

Date of Interview_____

EW System Represented_____

Section I : Projected Uses of EWNET

A. Resource Sharing

1. Peripheral Sharing = capability to access network from any terminal and access any peripheral in the network from any host.

2. File Access and Transfer = capability to transfer files between the devices and the hosts with all file restructuring transparent to the user.

3. Software Tool Sharing = capability to access programs, compilers, cross-assemblers, and utility routines anywhere in the network for the user's program.

4. Access To ARPANET = capability to access ARPANET from any terminal.

5. Access to AUTODIN I = capability to access AUTODIN I from any terminal.

6. Access To UNIVAC-1108 = capability to access the UNIVAC-1108 from any terminal.

7. Threat Generator = capability to use digital threat information generated from a central location.

8. Flight Test Monitor = capability to monitor Eglin AFB flight test in real time. **

B. Distributed Processing = capability of executing job processes that can run concurrently on different computers.

C. Distributed Databases = capability to access and maintain databases that are distributed across several computers in the network (Central Threat Database).

D. Video = capability to monitor Eglin AFB test video (ground and in-flight). **

E. Fault-Tolerance = capability to provide a more graceful degradation of user service when a network failure occurs.

F. Rate the Above Projected Uses.

PROJECTED USES	VERY GOOD	GOOD	MEDIUM	LITTLE USE	NO USE
PERIPHERAL SHARING	5	4	2	1	0
FILE TRANSFERS	8	2	1	1	0
SOFTWARE TOOL SHARING	8	3	1	0	0
ACCESS TO THE UNIVAC-1108	5	4	1	1	2
ACCESS TO AUTODIN I	1	1	2	0	8
ACCESS TO ARPANET	0	0	0	4	8
THREAT GENERATOR	4	5	1	3	1
FLIGHT TEST MONITOR	2	1	3	3	3
DISTRIBUTED PROCESSING	2	3	4	3	0
DISTRIBUTED DATABASES	2	3	4	4	0
VIDEO	1	2	3	2	4
FAULT-TOLERANCE	4	7	2	0	0

(Composite of User Responses)

G. What group of things would you like to see implemented first?

1. Peripheral Sharing_____4

2. Software Tool Sharing___1

3. File Transfer_____1

4. UNIVAC-1108 Access_____1

5. Distributed Processing___1

H. Prioritize the above group of uses with related comments:

1. File Transfers

2. Software Tool Sharing

3. Access to the UNIVAC-1108 - Backup of ISSes, Common Databases, and Off-Line Processing.

4. Peripheral Sharing - Critical peripherals such as plotters and printers.

5. Fault Tolerance - No support from DECNET.

6. Threat Generator

7. Rest of List in any order.

** Interface to Eglin AFB TM-Stations and Math Lab has not been defined. Possible satellite link to Eglin AFB is being considered.

Section II : Functional Requirements of EWNET

A. User-Oriented Requirements

1. Throughput

A. What computer / models do you envision using with your ISSes over the next 5 years?

B. What will be the average utilization in hours per day?

C. What will be the maximum utilization in hours per day?

COMPUTER	NORMAL USAGE	MAXIMUM USAGE	
UNIVAC-1108	8	24	
VAX 11/780	6	12	
VAX 11/780	4	7	
VAX 11/780	7	8	
VAX 11/780	10	24	
VAX 11/780	5	8	
VAX 11/780	8	24	
VAX 11/780	8	24	
VAX 11/780	8	12	
VAX 11/780	5	12	
PDP 11/34	6	12	
PDP 11/34	6	24	
PDP 11/34	6	24	
PDP 11/34	8	12	
D. G. ECLISPE	5	12	TEMPORARY
HARRIS 6024/4	4	24	TEMPORARY

2. Response Time

Do you forsee any projected uses of the network that requires that the response time of the network to a set of commands not exceed some threshold?

NO----------6

YES--Fast enough so no data is lost to a low sampling rate.

YES--Response time due to resource conflict should never exceed 5 seconds.

YES--Interactive terminals = 5 seconds.

YES--Interactive terminals < 1 second.

YES--File transfer ISS-to-ISS < 5 minutes.

3. User Interface

A. Would symbolic device accessing be desirable?

(Making the actual host that an I/O device is connected

to transparent to the user.)

YES----------13

NO---------- 0

OPTION------- 4

B. Should the same be done for software tools?

YES----------13

NO---------- 0

OPTION------- 4

C. Do you forsee the need for ISS / network interface

accounting data?

YES----------11

NO---------- 3

NOTE:

Number of times the ISS is externally accessed.

D. What other features should the network present

to the user?

Built-in-test---------- 1

User prompts and 'help' routines--------- 3

ISS independence----------13

In-house distribution

Point-to-point---------- 3

Broadcast---------- 1

4. Security

A. Do you forsee the running of classitied data or programs on the network, and if so at what classification?

YES----------14

NO---------- 0

SECRET

B. Do you forsee the need to protect databases on the network from unauthorized access?

YES----------14

NO---------- 0

COMMENT:

2 Element-password to protect need-to-know Security accountability = if the net could record time, destination, and password of classified receiptents, then would be a benefit to audits of activity when problems arise.

5. Availability

A. What is the minimum percentage of time that you

feel that the network should be available?

B. What is the reason for giving the above availability?

100 %-----6 (0800-1700 hours + emergency changes 5 day work week)

99.9 %----1

95 %-----2 (24 hour day, downtime = 5 %)

90 %-----1 (network required for full ISS capability)

50 %-----2

C. Special consideration for availability: Emergency changes should have priority when field support is required, nothing should get in the way.

6. ISS Interaction

How and why do you envision ISS-to-ISS interaction?

1. Internal to section only---------- 5
2. No external / limited external----------10
3. Interaction with simulation & analysis------ 1
4. Interaction with Flight Line Tests Sets----- 1

7. Other User-Oriented Requirements?

1. Common network command language----------2
2. Central documentation----------2
3. Word-processor----------5

4. Data link to Elint sources----------1

5. Overflow / surge load handling----------1

   (Using another ISS's terminals

   to operate your own ISS)

6. Self-configuring through periodic

   status checks of the network members---------1

7. Real-time capabitity where one

   ISS takes complete control of network--------2

B. How would you rank the functional requirements for the above areas?

AREA	NOT APPLY	MARGIN	APPLY	VERY APPLY	MUST HAVE
THROUGHPUT	0	0	3	7	3
RESPONSE TIME	0	1	2	7	3
USER INTERFACE	0	0	2	4	8
SECURITY	0	0	0	0	12
AVAILABILITY	0	0	4	4	5
ISS-TO-ISS INTERACTION	4	4	1	2	1

(Composite of User Responses)

Section III : Other Comments

What other comments or suggestions do you have concerning the projected uses and/or functional requirements of EWNET?

1. The network software must be interrupt-driven. (Undecided)

2. It would be nice to be able to interrogate the network to find a file that was sent. (6)

3. The network should not be used for general software development. (4)

4. It should be easy to initialize the network with an arbitrary subset of the nodes available on the network. (4)

5. The number of nodes and their locations should be addressed. (4)

6. The number of ISSes per node should be addressed. (4)

7. Fiber optics should be used for the interface cables. (4)

8. The network must be designed for a high level of interaction for future growth but must be capable of implementation with very limited capabilities. (13)

Section IV : Design-Oriented Functional Requirements

The following requirements are not user-oriented and should be answered by section and branch management. Section and branch management are being asked to answer the questions in this section since no single network management group presently exists within the Engineering Branch.

Management Title_____

A. Design-Oriented Requirements

1. Flexibility

A. What changes do you see being made to the network during the next few years?

1. More hosts and devices being added----------6

2. Being used more for non-EW purposes----------3

3. Workload changing from mainly file transfers-----3

4. More software development systems added----------5

5. Expanded use of on-line threat database----------1

6. Usage expansion as value is identified----------1

B. How important do you feel it is for EWNET to be easily reconfigurable with respect to the following?

NETWORK SUBCOMPONENTS	VERY	SOME	NOT
NEW COMPUTERS AND DEVICES	7	3	0
NEW TOPOLOGIES	2	3	5
NEW PROTOCOLS	2	2	6
NEW TRANSMISSION MEDIUM	2	3	5
SERIAL-TO-PARALLEL	1	7	2

(Composite of User Responses)

2. Performance Monitoring

What performance monitoring capabilities should EWNET have?

1. Collect accounting data----------9

2. Collect node statistics----------7

3. Hardware monitors----------1

4. Software monitors----------7

5. Monitor network bottlenecks----------9

6. Monitor network status----------9

7. Performance monitoring node----------3

NOTE:

To protect from network slowdown must be on demand basis only.

3. Distributed Processing Language

What language(s) would you like to see implemented on all or most of the hosts if a distributed processing capability is implemented?

1. Jovial J-73--------- 4

2. Pascal---------- 3

3. ADA----------10

4. Fortran---------- 6

5. Basic---------- 5

NOTE:

ADA has powerful control structures for distributed processing.

4. Other Design-Oriented Requirements

Are there any other design oriented requirements that should be addressed?

1. System should contain dual pass-word access.

2. Infinite queue detection

B. How would you rank the functional requirements for the above areas?

AREA	NOT APPLY	MARGIN APPLY	APPLY	VERY APPLY	MUST HAVE
FLEXIBILITY	0	0	2	3	4
PERFORMANCE MONITORING	0	0	4	3	2
DIST. PROCESS LANGUAGE	0	0	6	1	2

(Composite of User Responses)

C. General Comments

Al Richardson: "I see the host processor group handling the nodes and documentation, word processing, backup requirements for all ISSes connected to the network. The software common tools and the database must be resident at nodes common to all systems. The EWOLS and ECSAS systems must be able to operate through the network from any CRT terminal and not interfere with ISS operation".

Everyone: "Don't want own ISS tied up by someone

else. Each ISS should process its own jobs first before servicing anyone else".

Management: "If it is possible, we world like to see the functions performed by the UNIVAC-1108, performed by the Network Node Computers."

Section V : List of Users Who were Interviewed

USER'S NAME	EW SYSTEM CONTROLLED	OFFICE SYMBOL
Mr. Joe Black	Electronic Warfare Branch Chief	MMRR
Mr. John LaVecchia	Simulation & Analysis Chief	MMRRA
Maj. Al Richardson	Unit Chief	MMRRAA
Mr. Russell Decote	Standardization	MMRRAA
Mr. Jerry Nachreiner	ECSAS	MMRRAA
Mr. Don Schroeder	EWOLS	MMRRAA
Mr. Art Daum	UNIVAC-1108	MMRRAH
Mr. Bobby McDonald	Jammer Section Chief	MMRRC
Mr. Dean DeLaigle	ALQ-119 Unit Chief	MMRRCB
Mr. Dennis Thomas	ALQ-155	MMRRCC
Mr. H. J. Hildreth	Integrated Sys. Section Chief	MMRRI
Mr. Jerry Reynolds	F-15 TEWS	MMRRIA
Mr. Hal Maney	EF-111A Unit Chief	MMRRIC
Mr. Gary Cox	USM-464 (FLTS)	MMRRIC
Mr. Bert Goble	APR-38	MMRRIT
Mr. Tom Ridout	APR-38	MMRRIT
Mr. Tom Batterman	RWR Sys. Section Chief	MMRRV
Mr. Phil Oliver	ALR-46 Unit Chief	MMRRVA
Mr. Bob Smock	ALR-46	MMRRVA
Mr. John Louth	ALR-46	MMRRVA
Mr. N. Mitin	ALR-62	MMRRVB
Mr. Harry Jennings	ALR-69 Unit Chief	MMRRVC

Appendix B

Electronic Warfare Engineering Branch Laboratory Floor Layout Diagram and Topology Structure

This appendix contains a floor layout diagram of the Electronic Wartare Engineering Branch Laboratory at Robins AFB, Georgia. The locatıon of all ISS computers with semi-permanent locations are shown as well as the proposed locations of the EWNET node computers. Finally, the computer links (topology) that were initıally proposed by the Georgia Institute of Technology to connect the node computers are shown (Ref. 2:10).

Contents

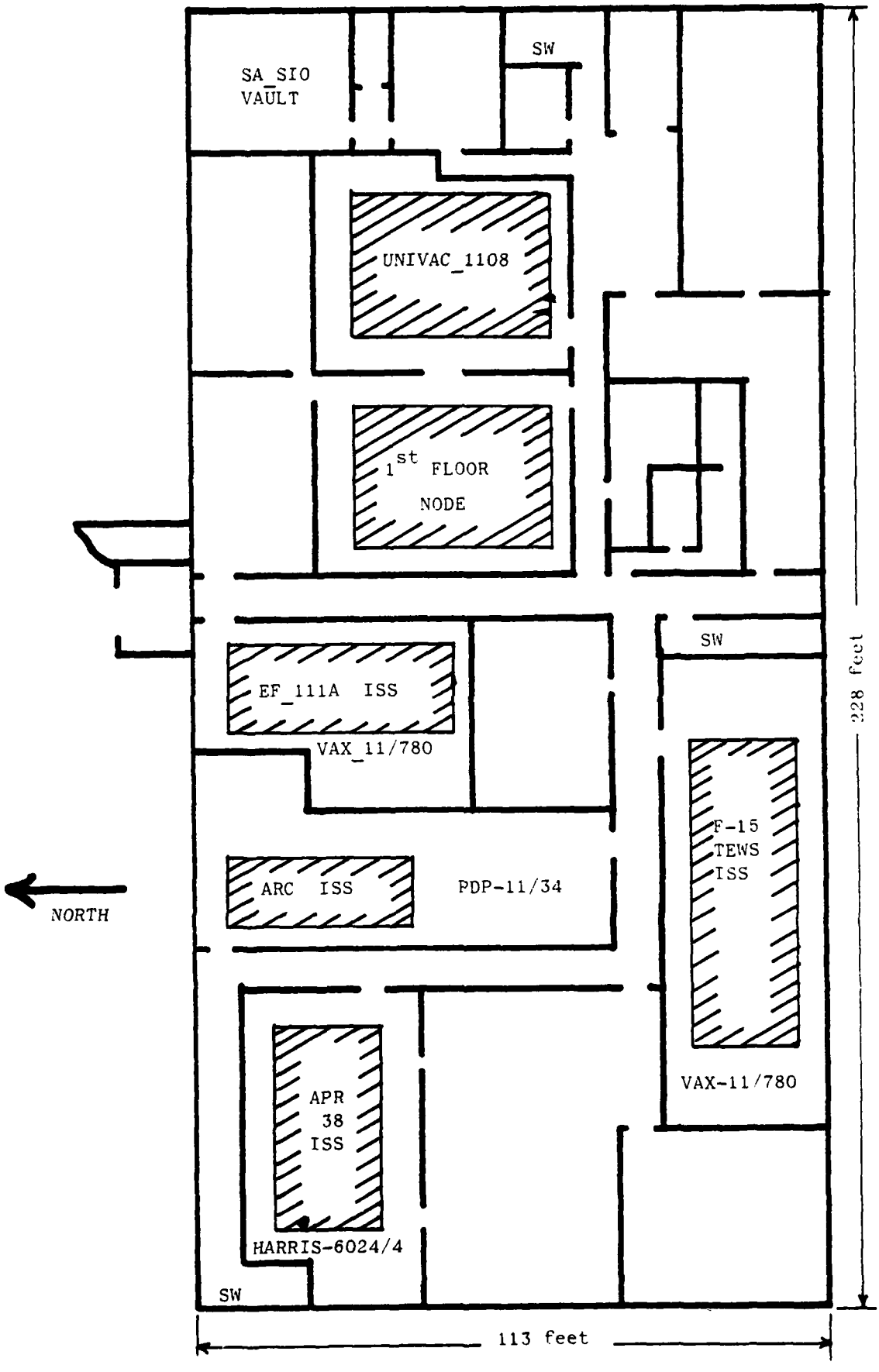


Figure 1-B 1st Floor Layout

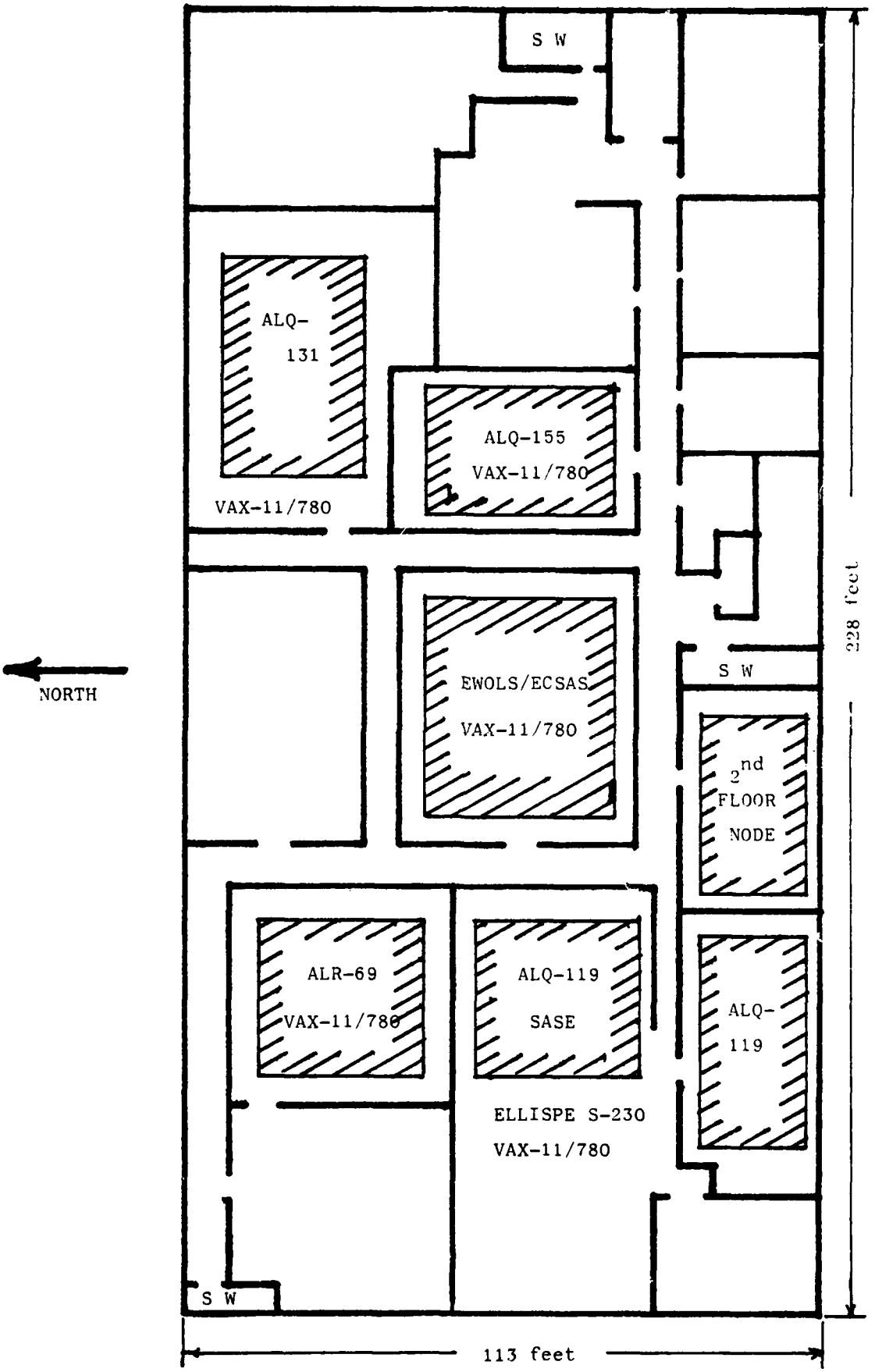


Figure 2-B.2 nd Floor Layout

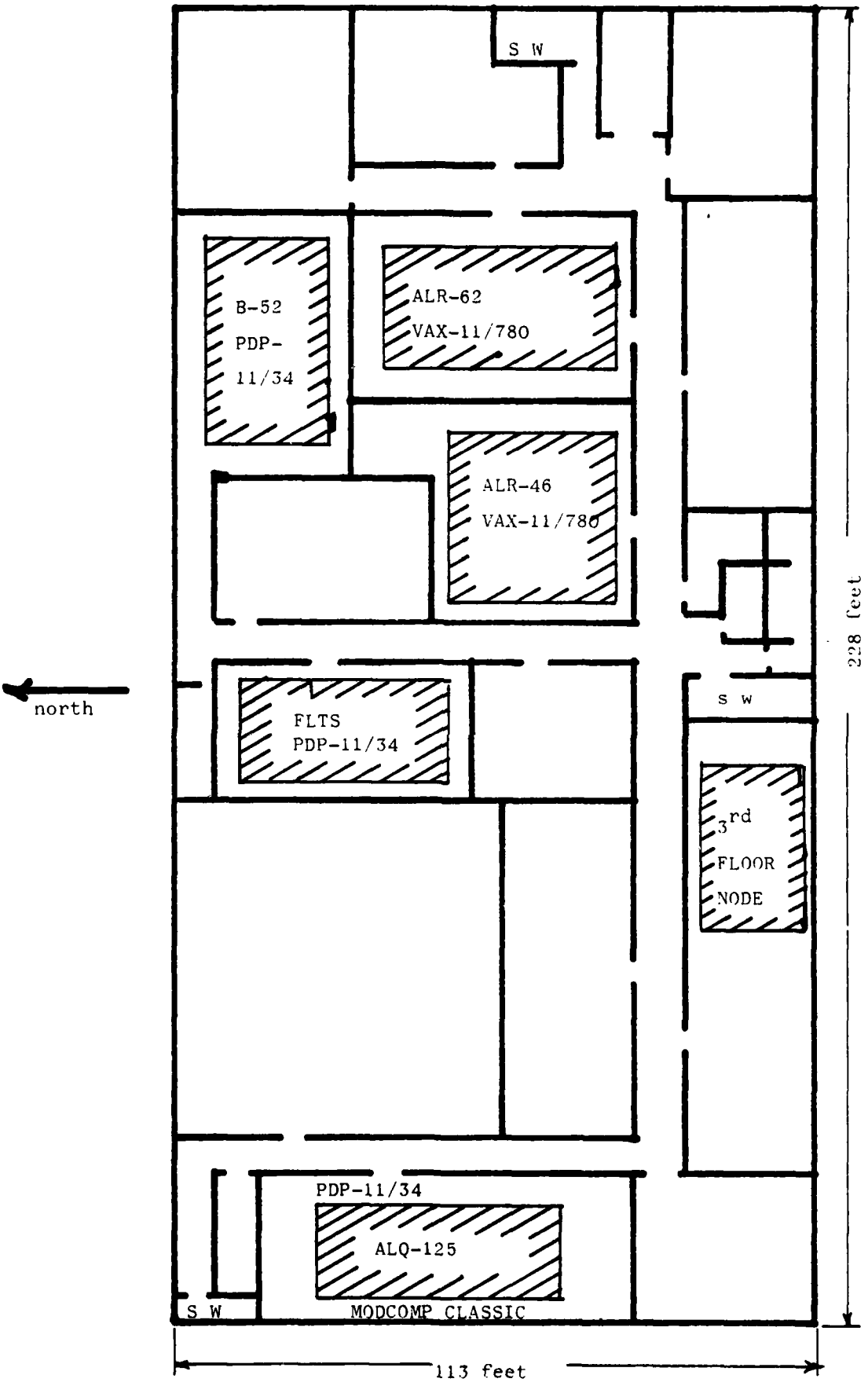


Figure 3- B 3rd Floor Layout

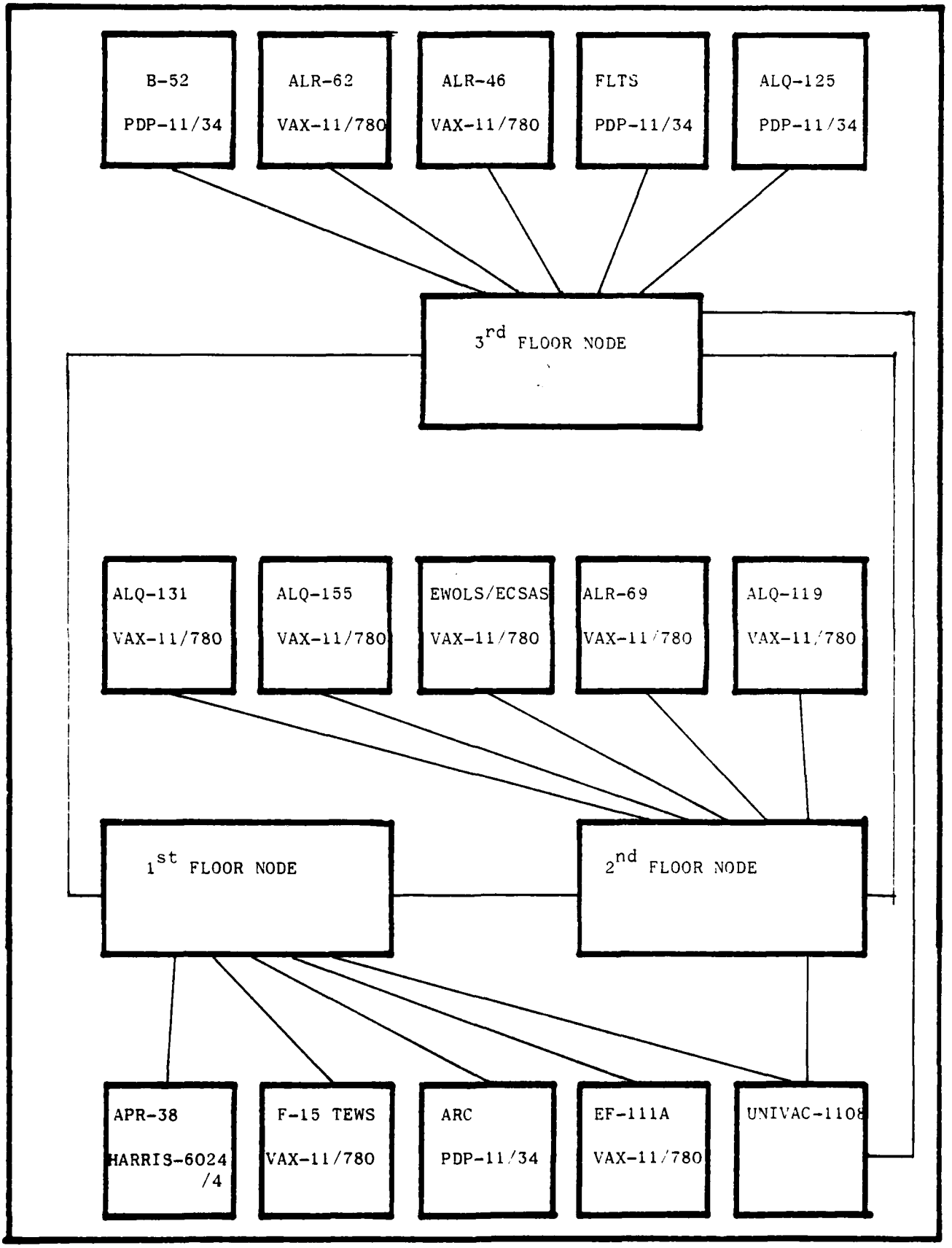


Figure 4-B Topology Structure

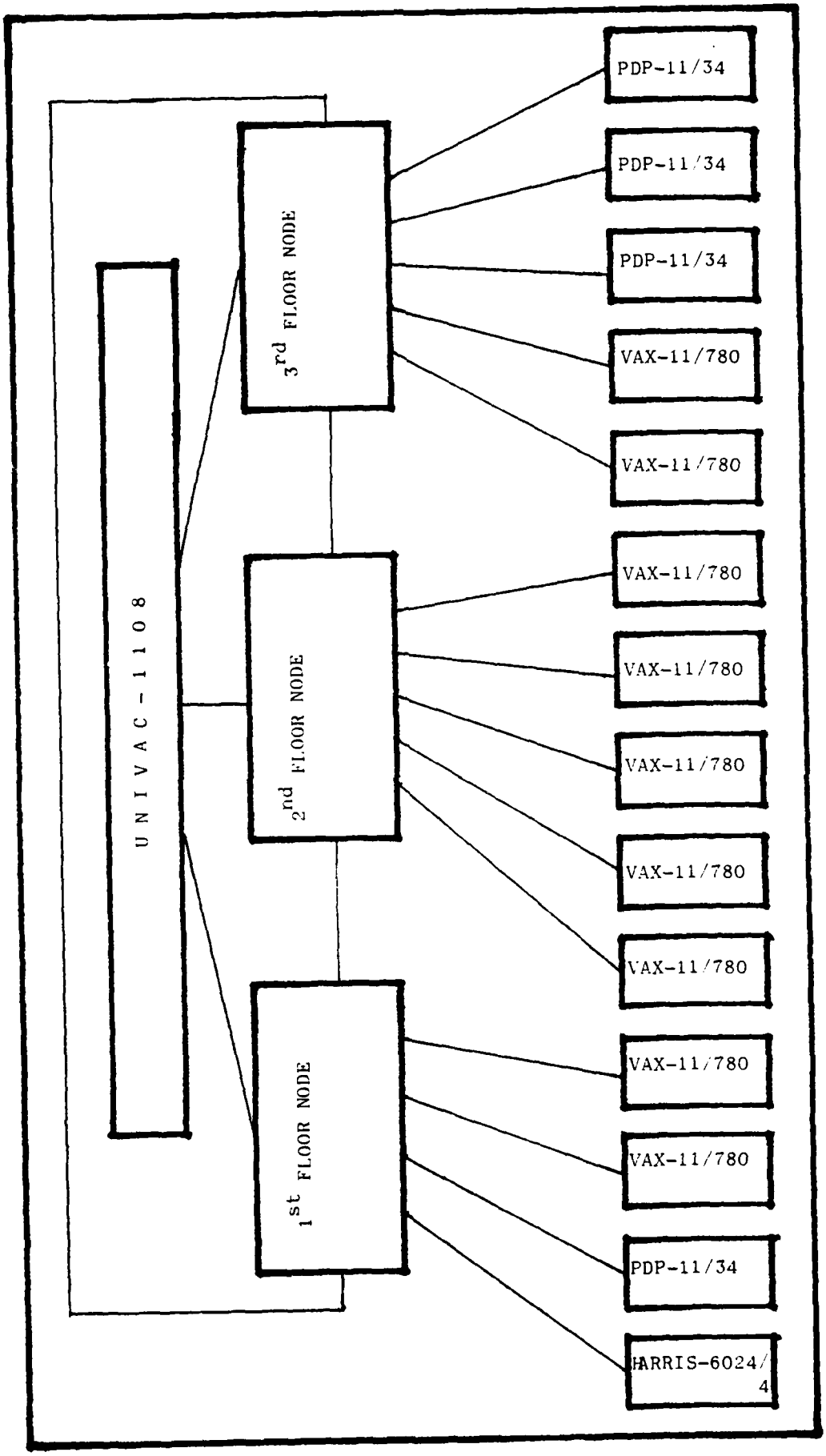


Figure 5-B Topology Structure

Appendix C

Structured Specification

This appendix contains the structured specification of the software requirements for EWNET. First, the complete set of data flow diagrams is included. These are followed by the data dictionaries for the high-level protocol (user-level), the architecture-level protocol, the transport/network-level protocol, and the link-level protocol.

Contents

Data Flow Diagrams for User Level Protocol

Contents

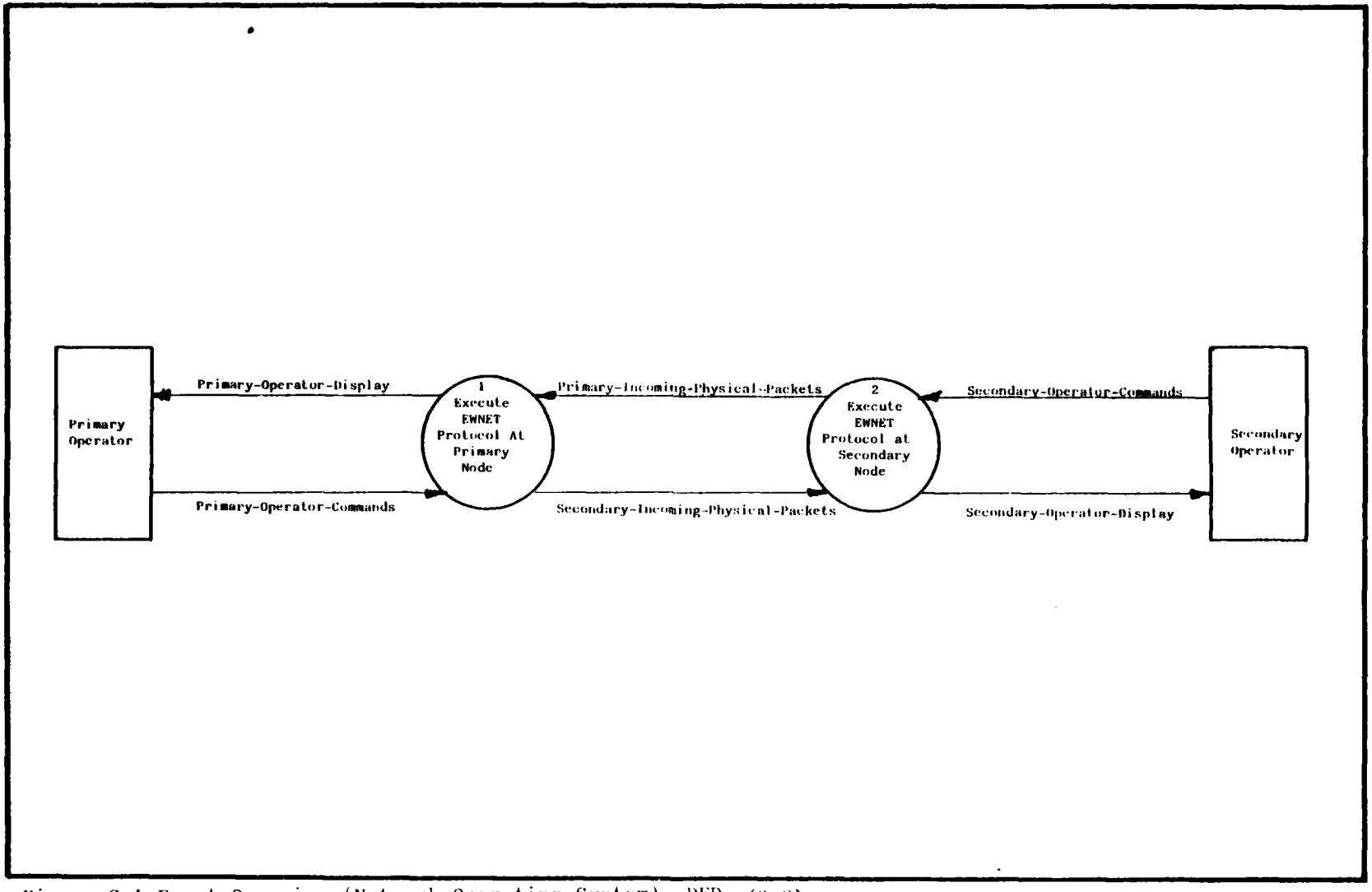


Figure C-1 Ewnet Overview (Network Operating System) DFD (0.0)

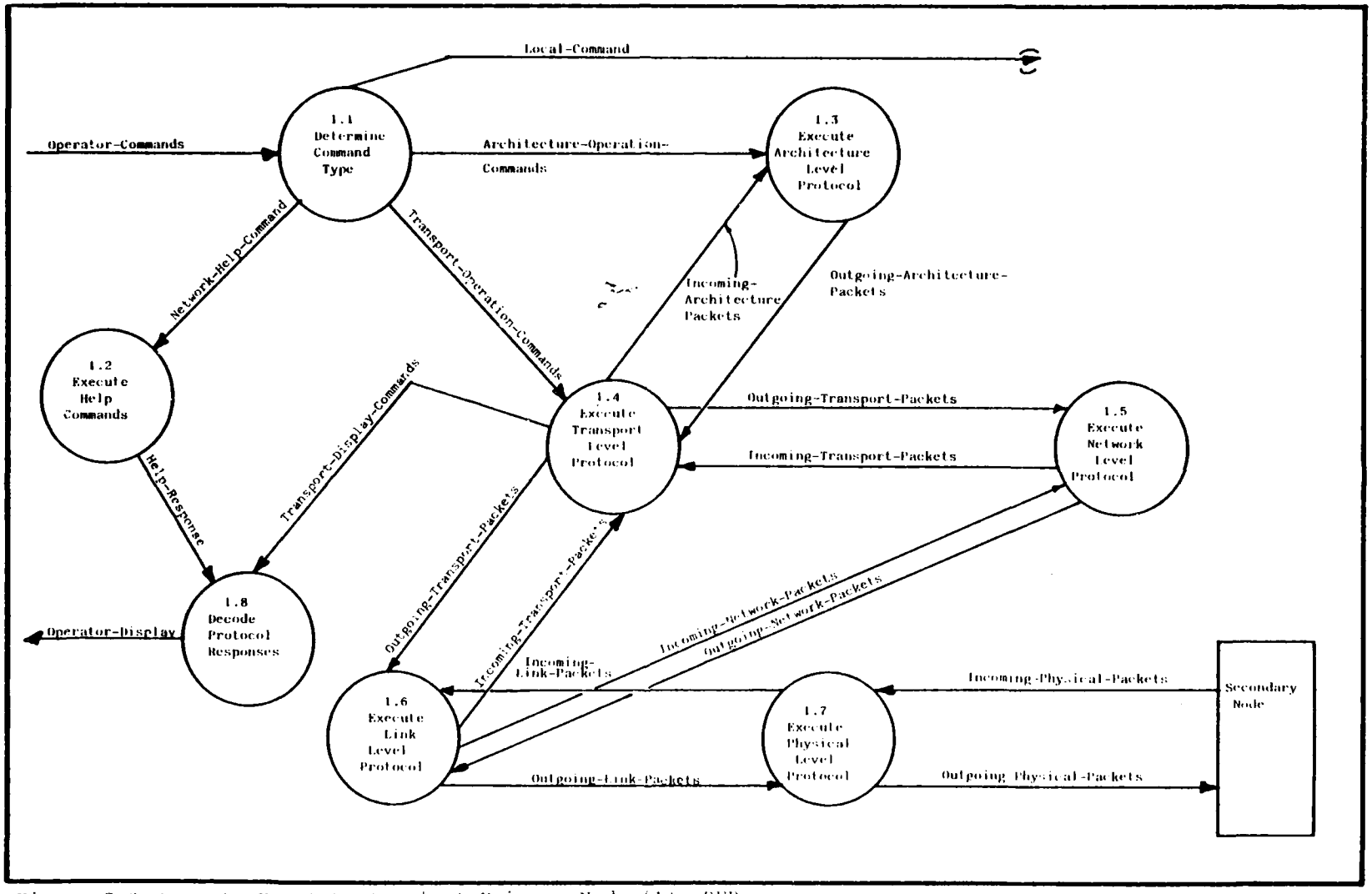


Figure C-2 Execute Ewnet Protocol at Primary Node (1) DFD

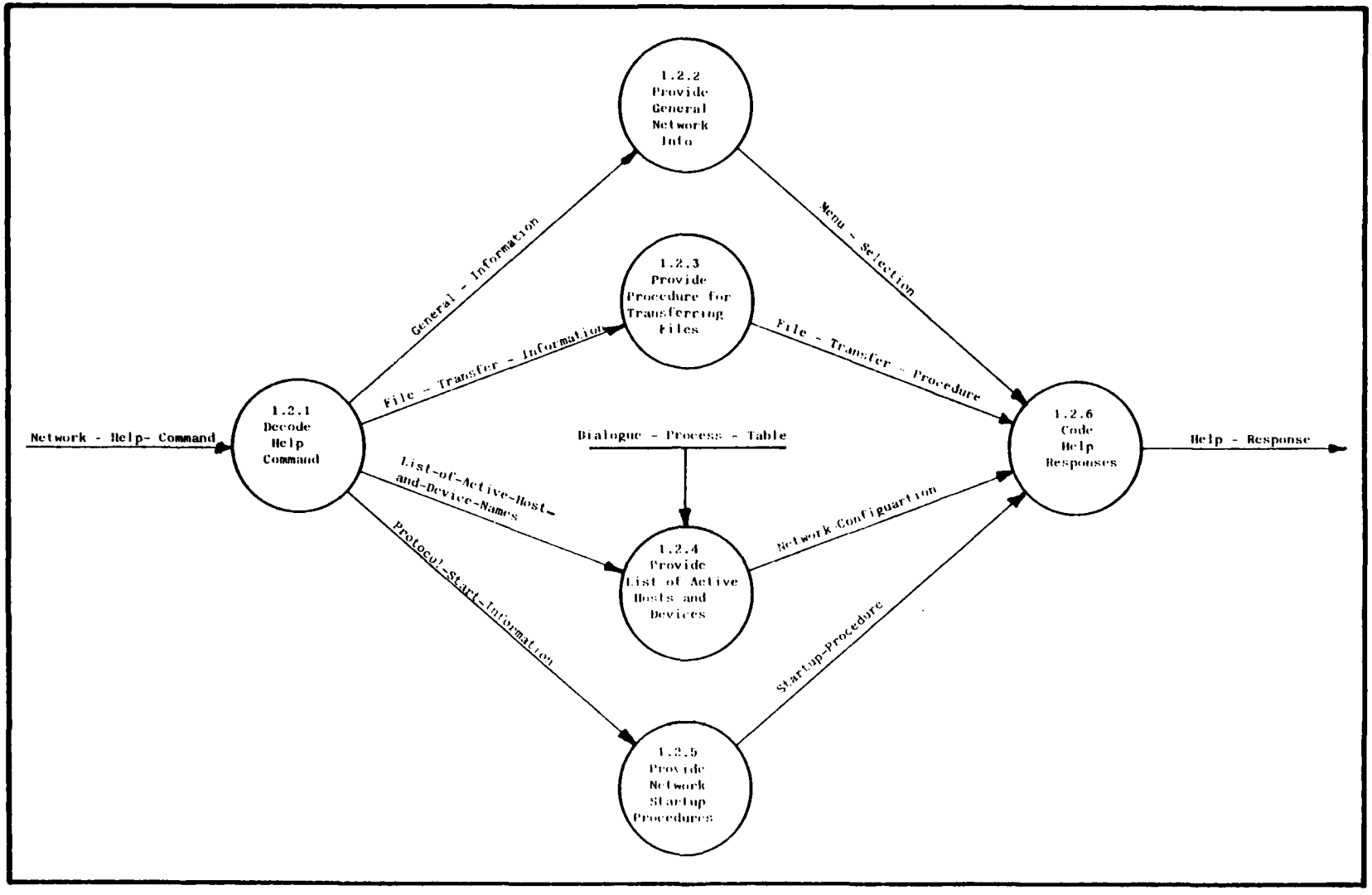


Figure C- Execute Help Commands (1.2) DFD

Data Flow Diagrams for Architecture Level Protocol

Contents

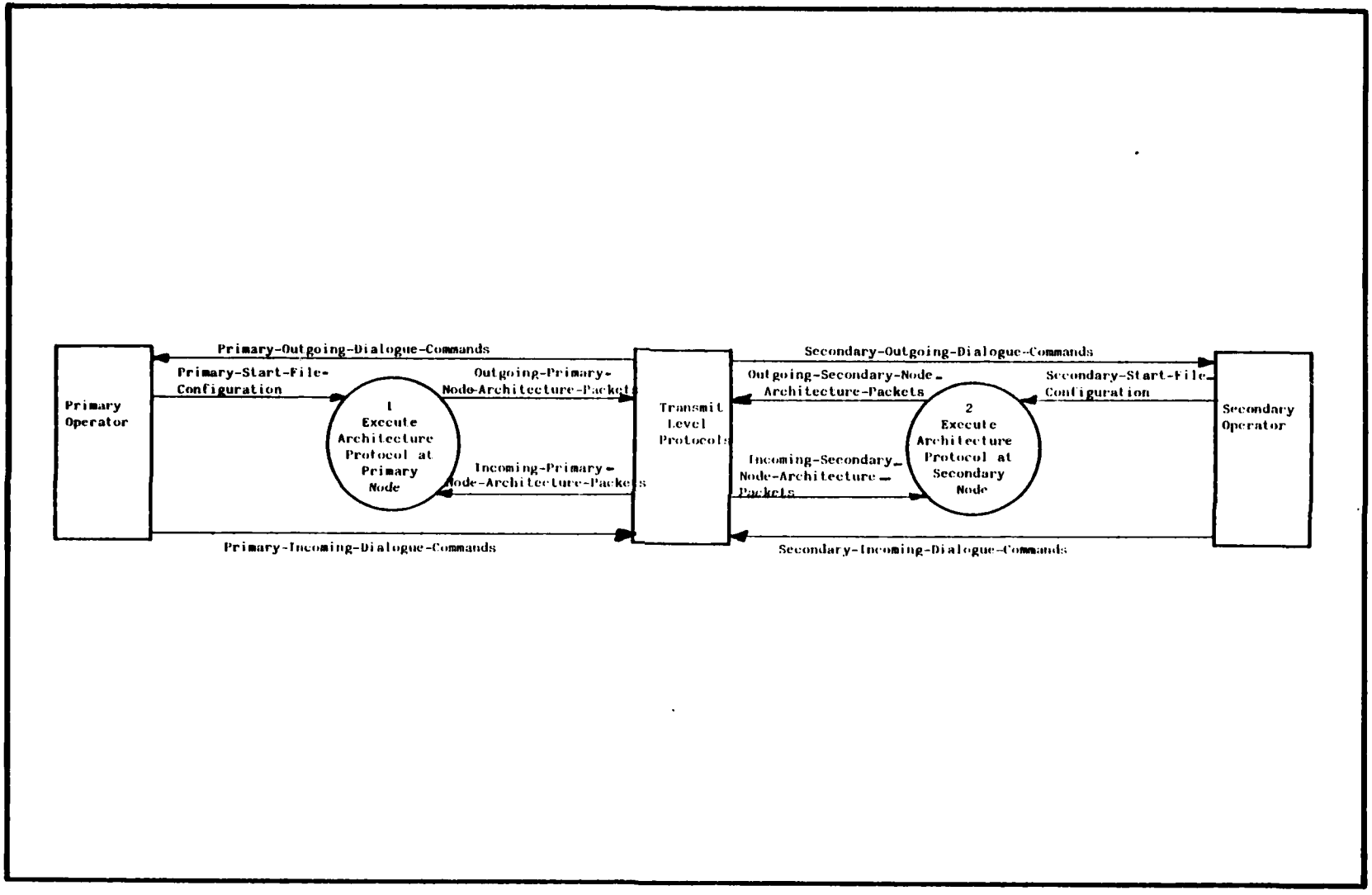


Figure C-4 Ewnet (Architecture) Overview DFD

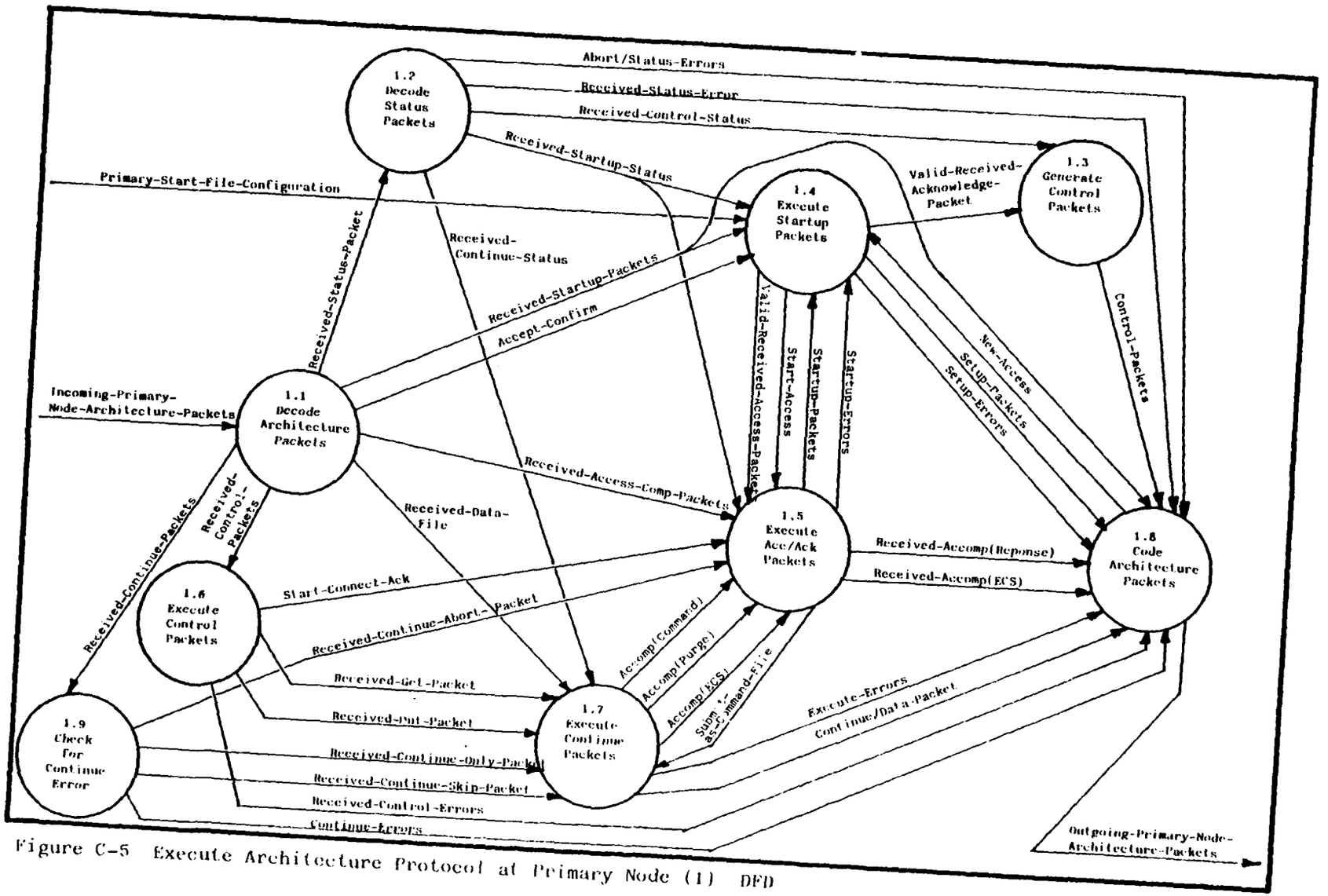


Figure C-5 Execute Architecture Protocol at Primary Node (1) DFD

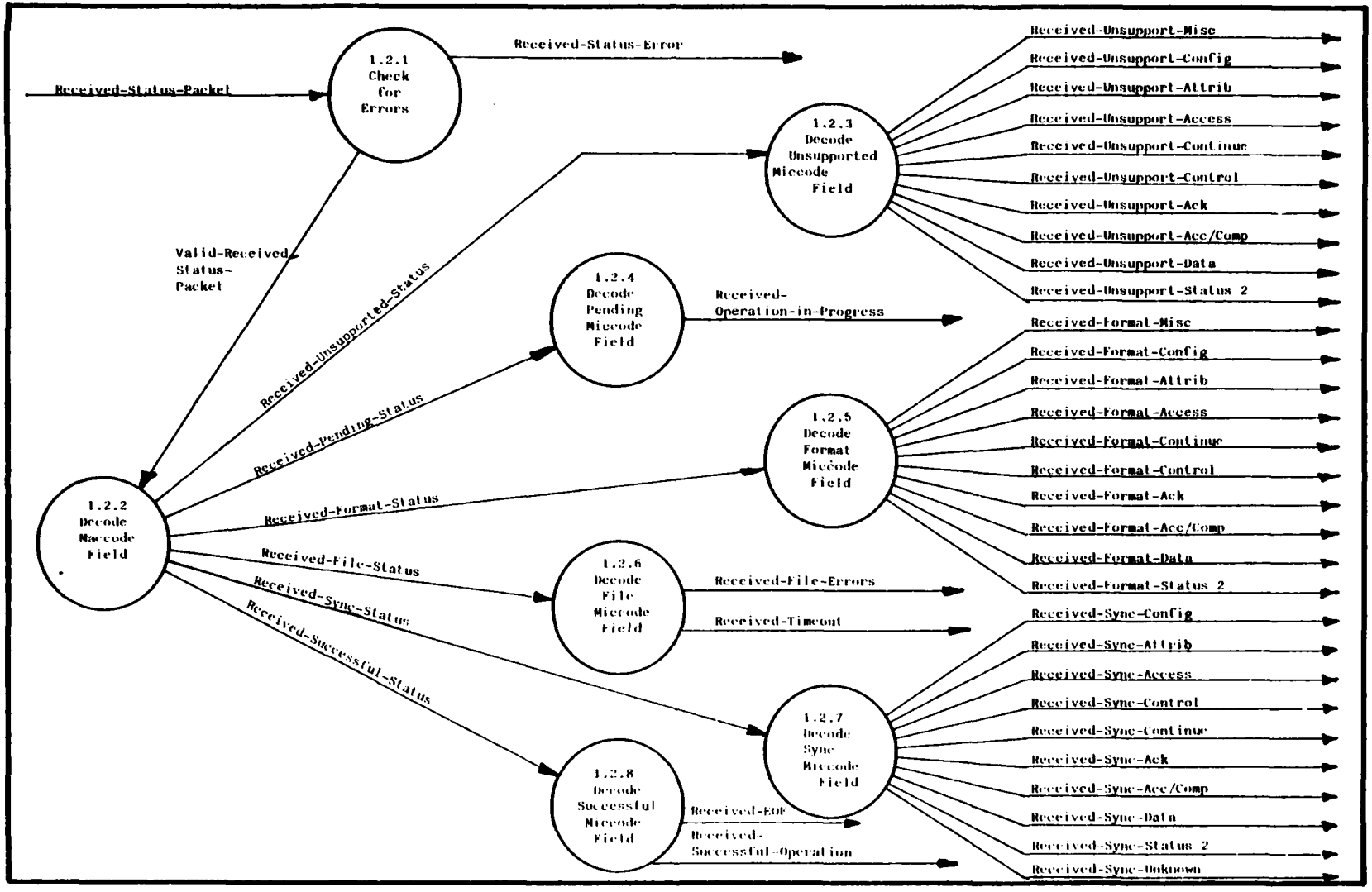


Figure C-6 Decode Status Packets (1.2) DFD

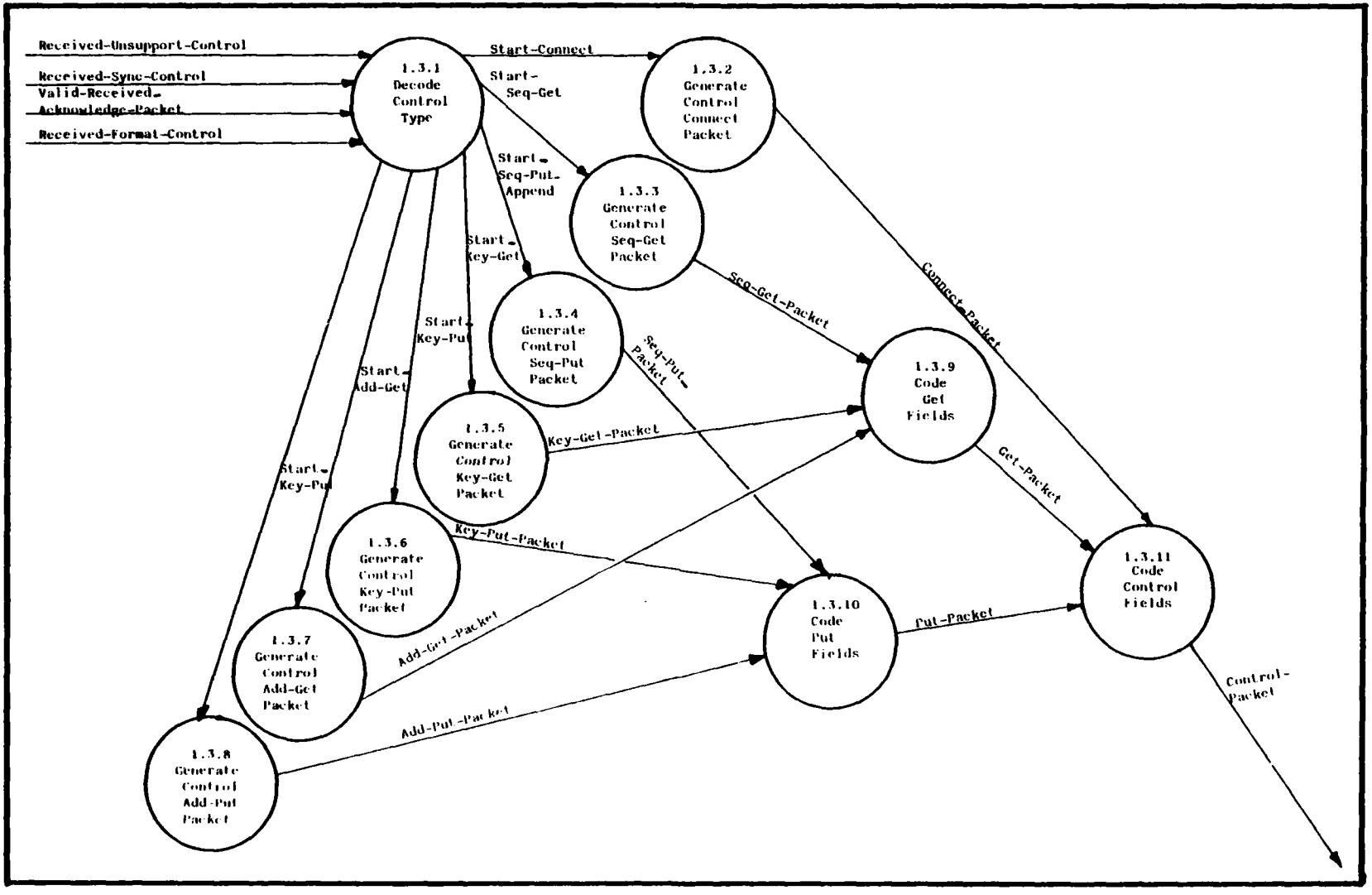


Figure C-7 Generate Control Packets (1.3) DFD

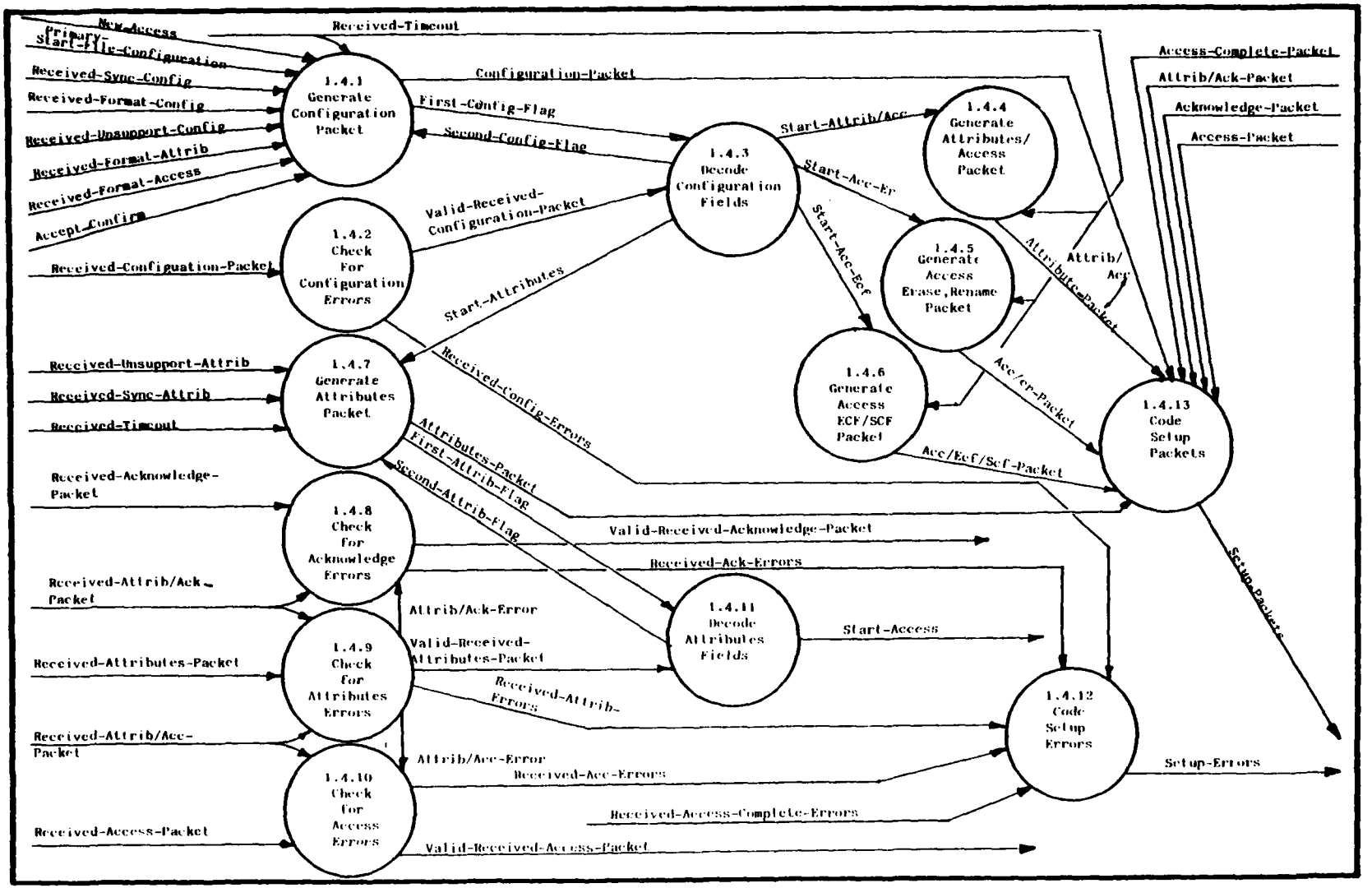


Figure C-8 Execute Startup Packets (1.4) DFD

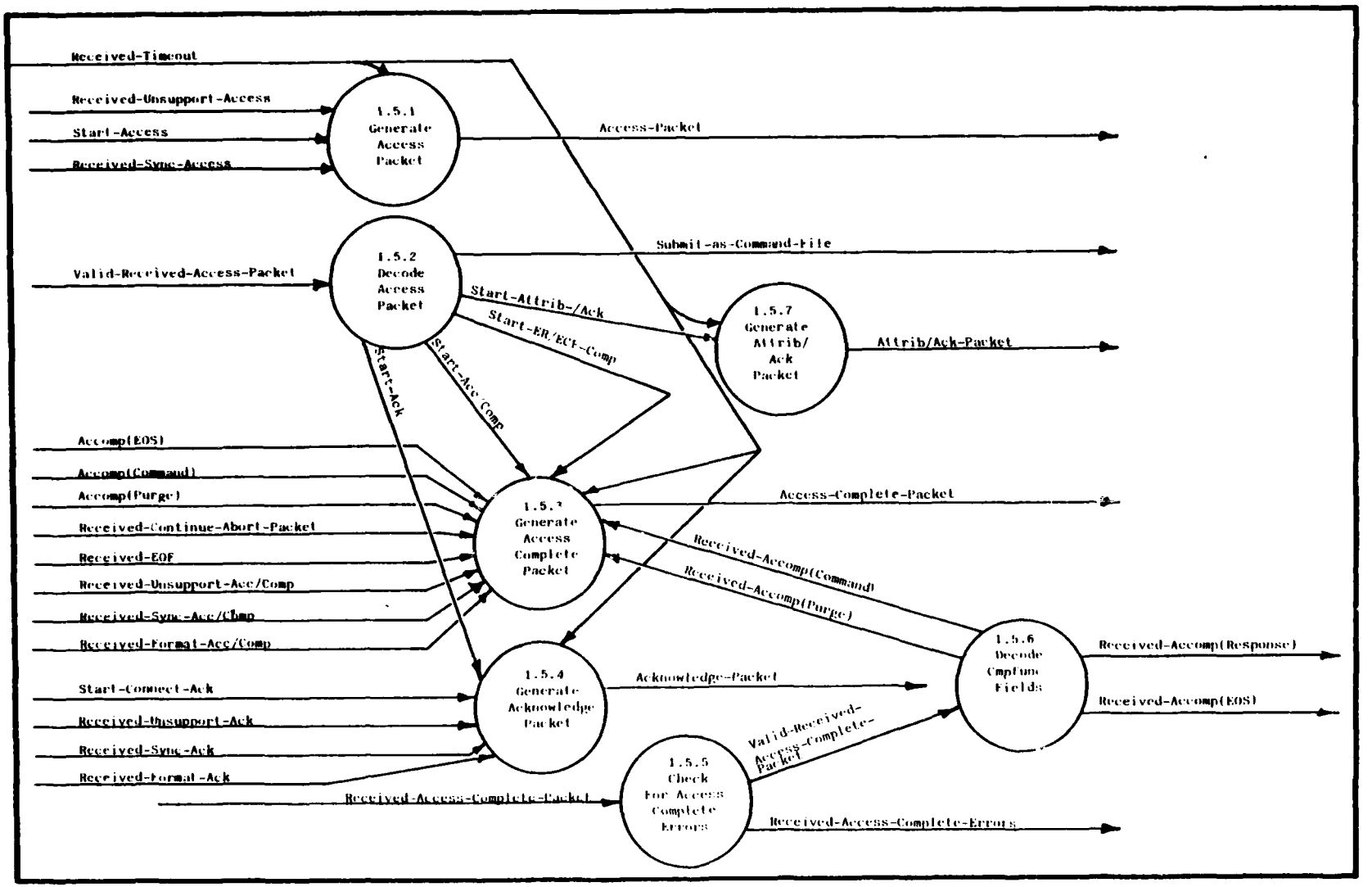


Figure C-9 Execute Acc/Ack Packets (1.5) DFD

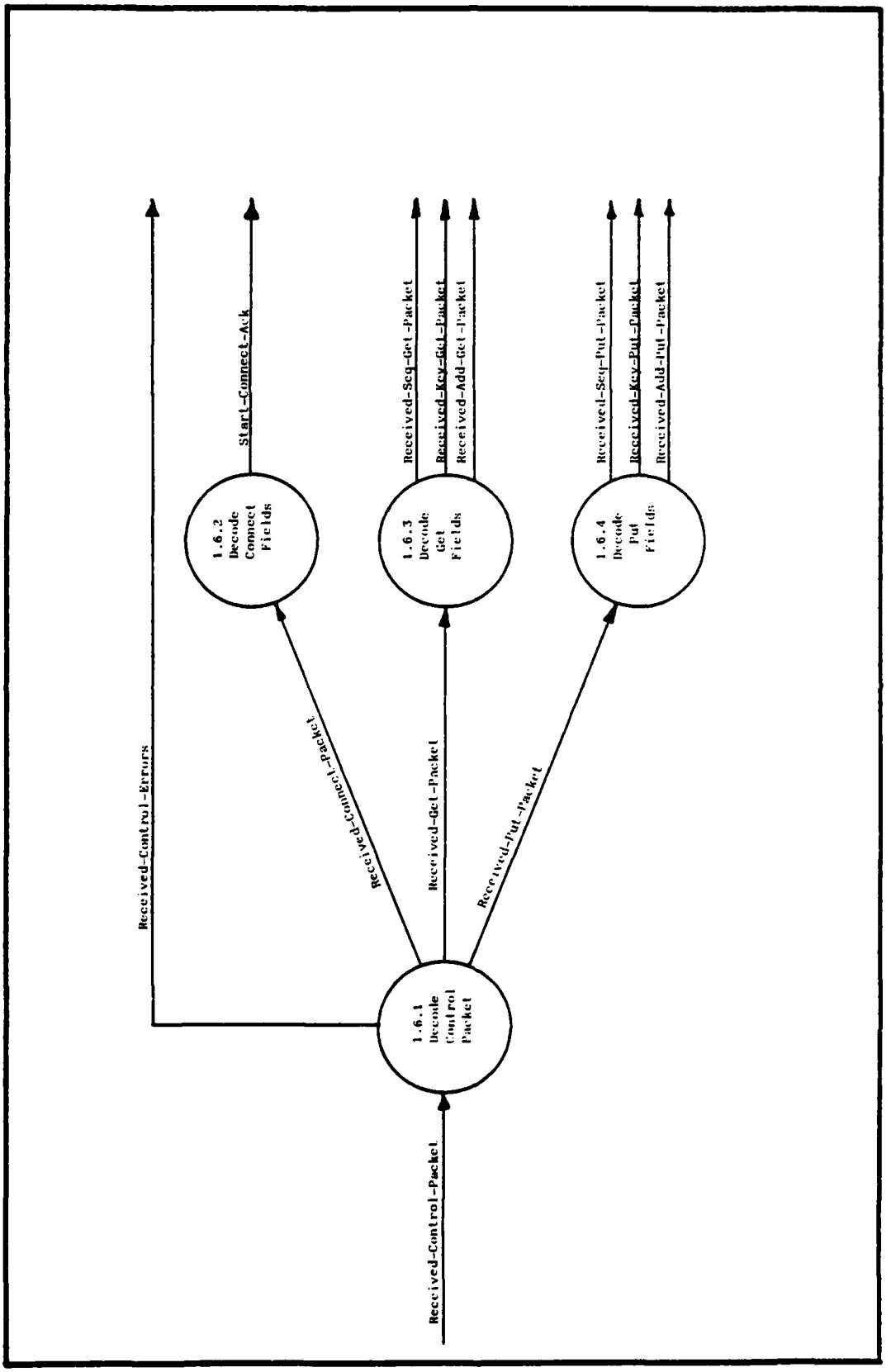


Figure C-10 Execute Control Packets (1.6) DFD

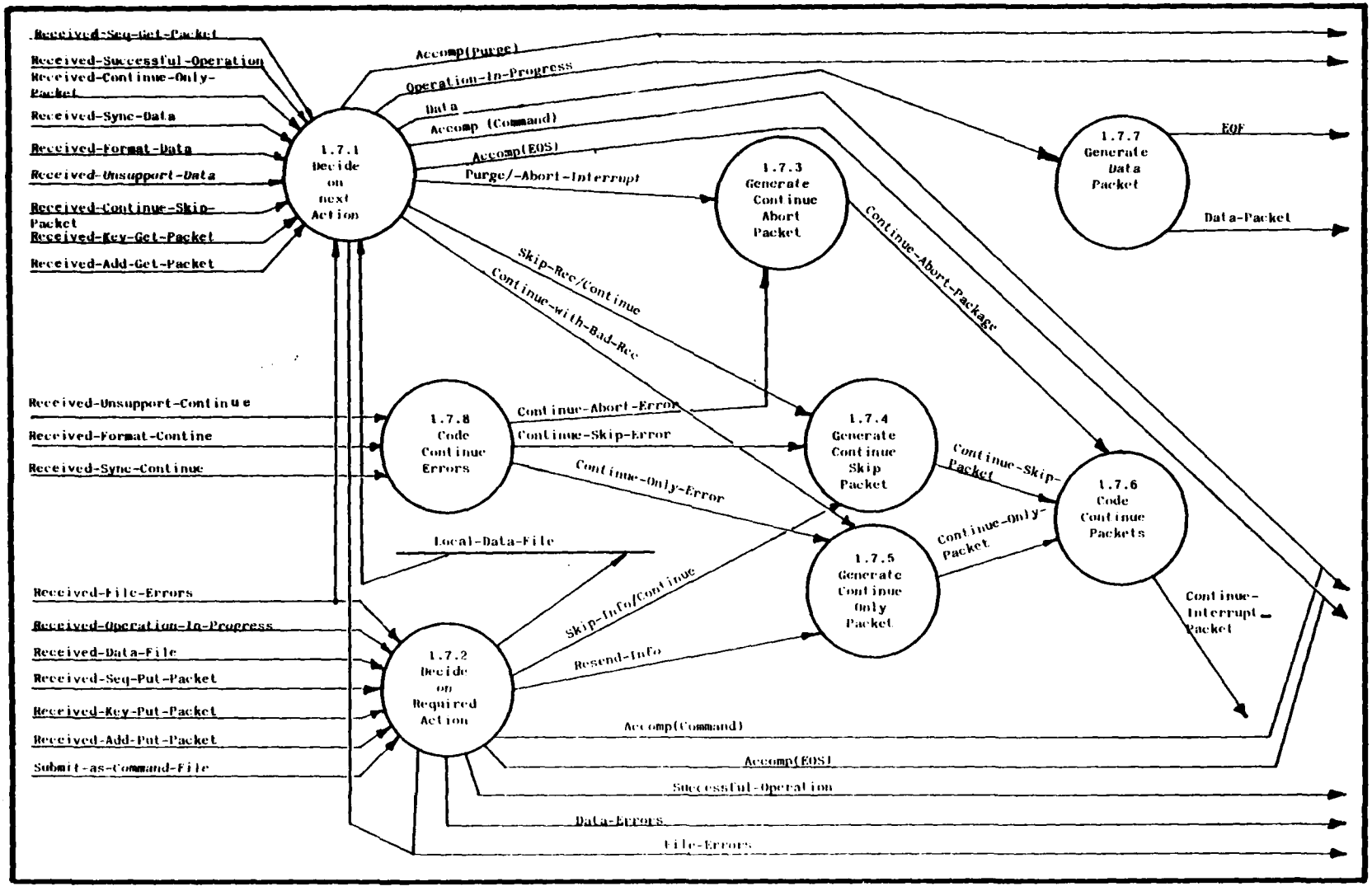


Figure C-11 Execute Continue Packets (1.7) DFD

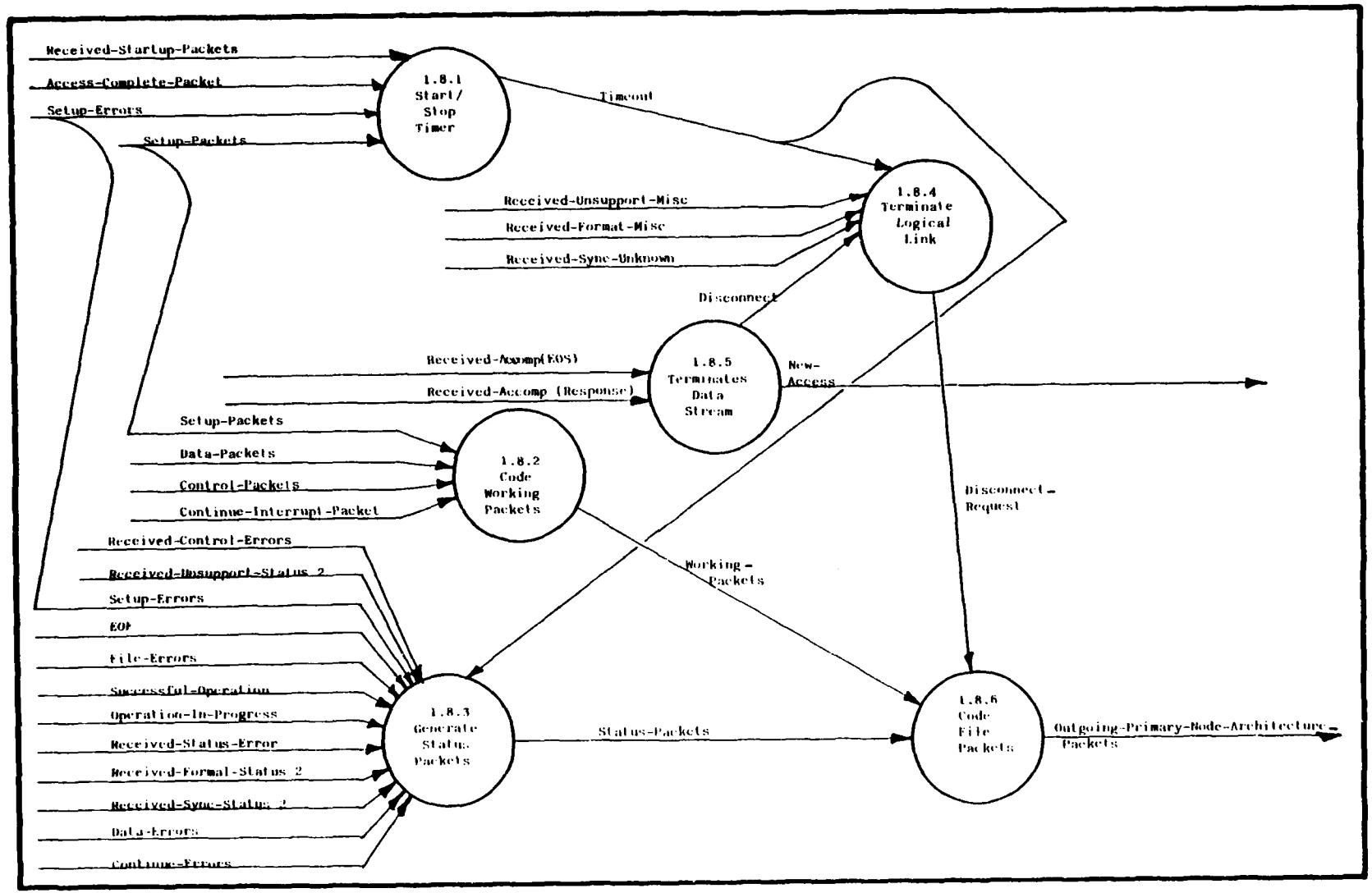


Figure C-12 Code Architecture Packets (1.8) DFD

Data Flow Diagrams for Transport/Network Level Protocol

Contents

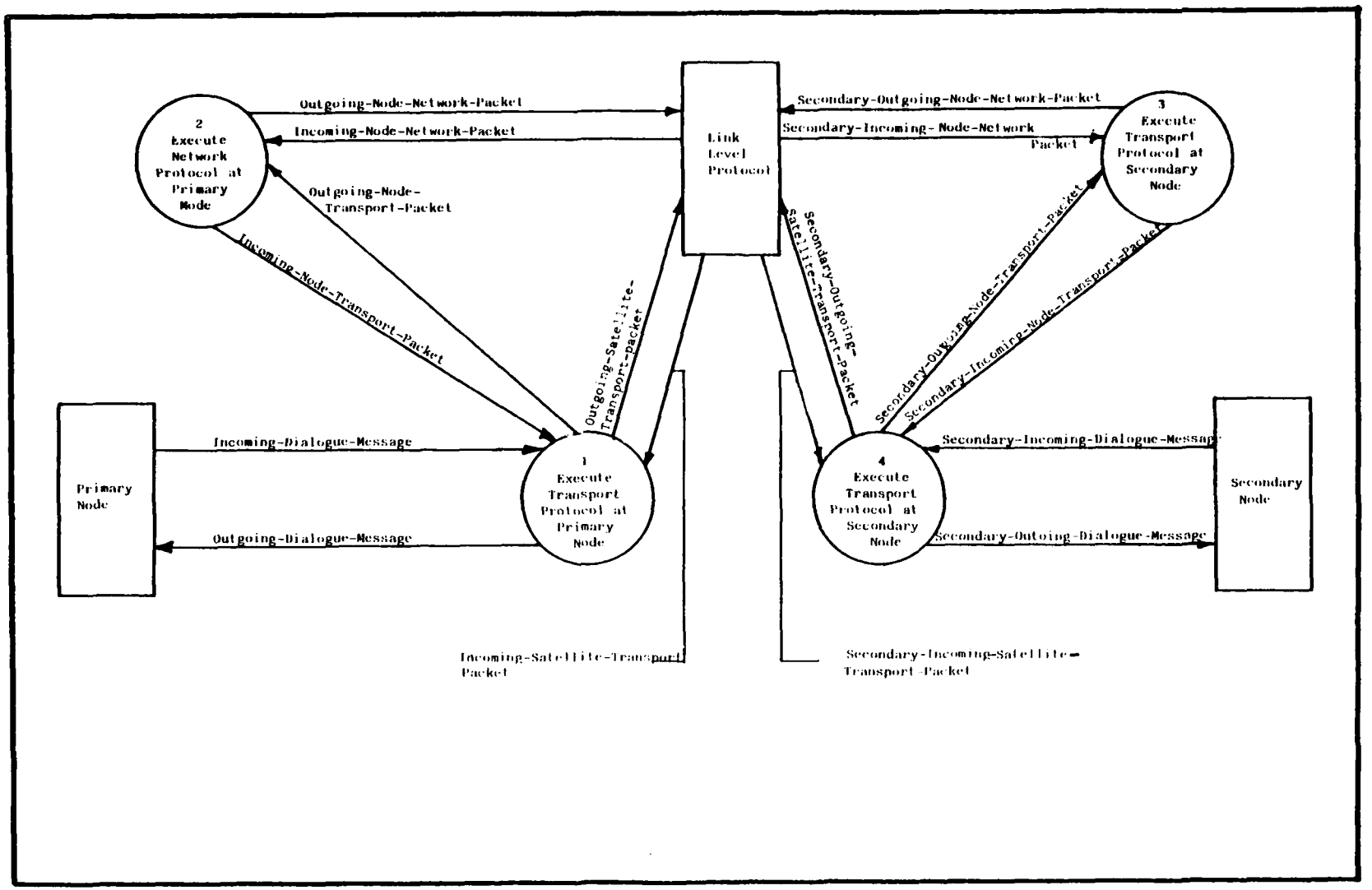


Figure C-13 Ewnet (Transport/Network) Overview DFD

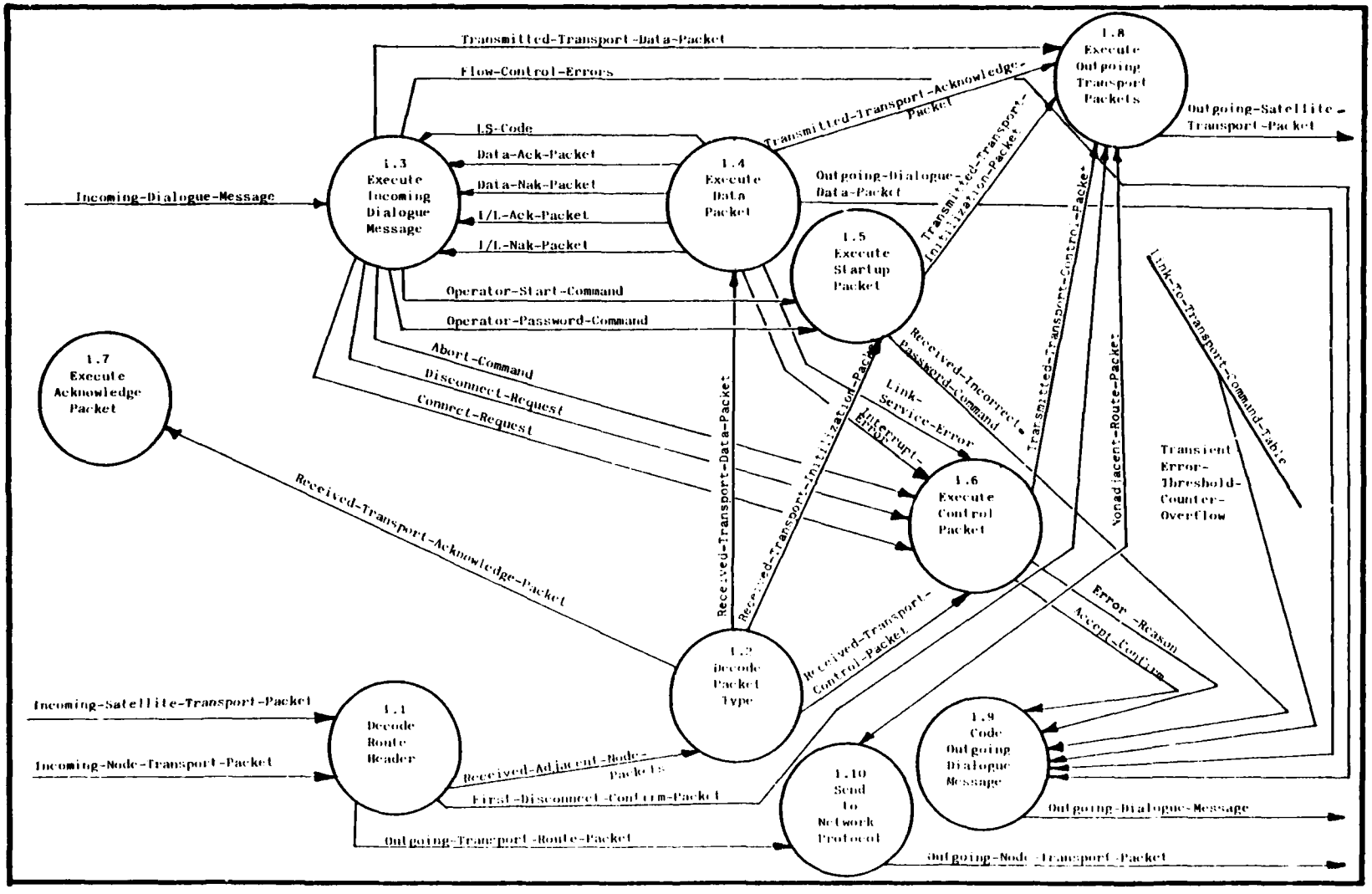


Figure C-14 Execute Transport Protocol at Primary Node (1) DFD

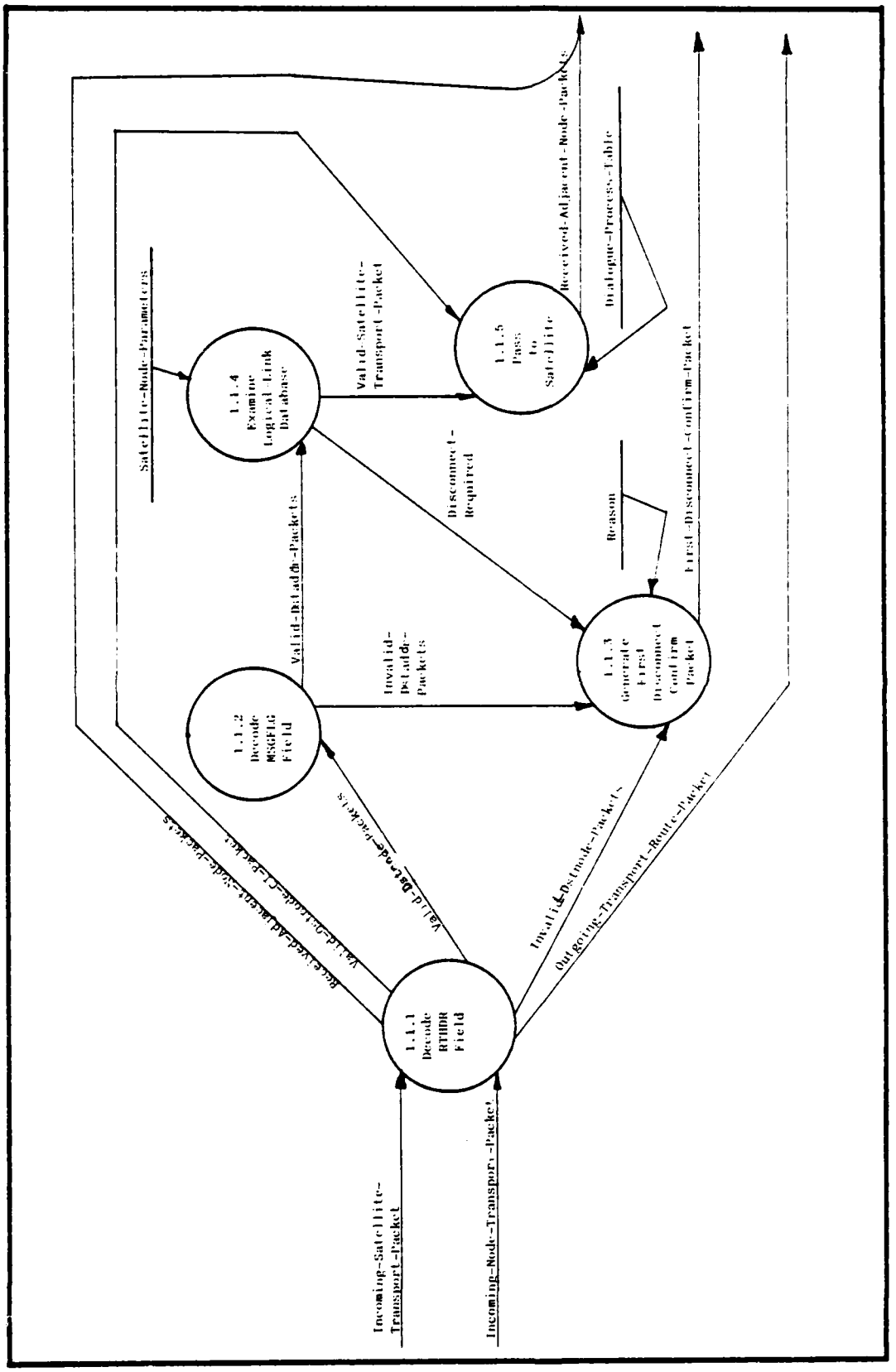


Figure C-15 Decode Route Header (1.1) DFD

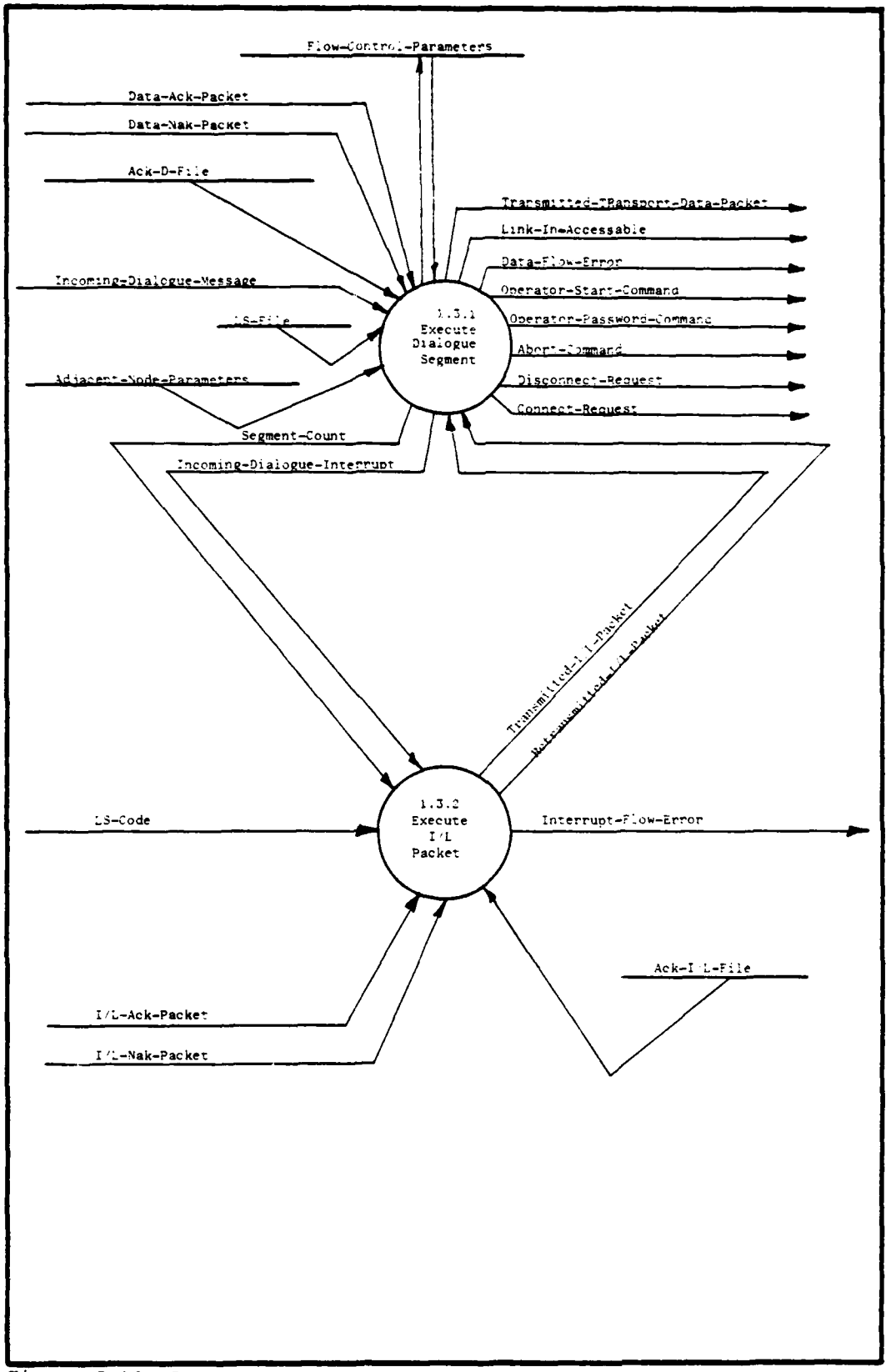


Figure C-16 Execute Incoming Dialogue Message (1.3) DFD

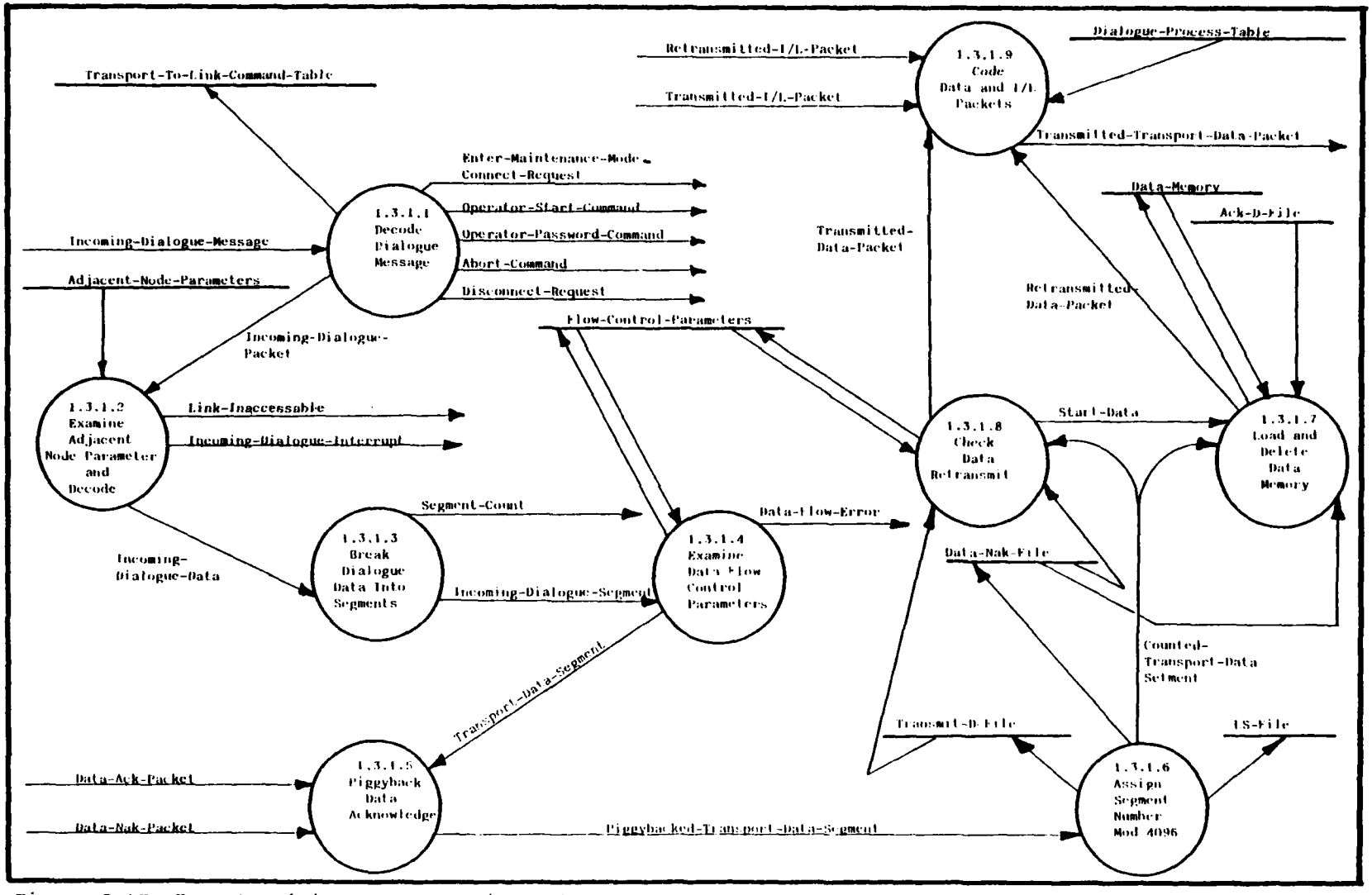


Figure C-17 Execute Dialogue Segment (1.3.1) DFD

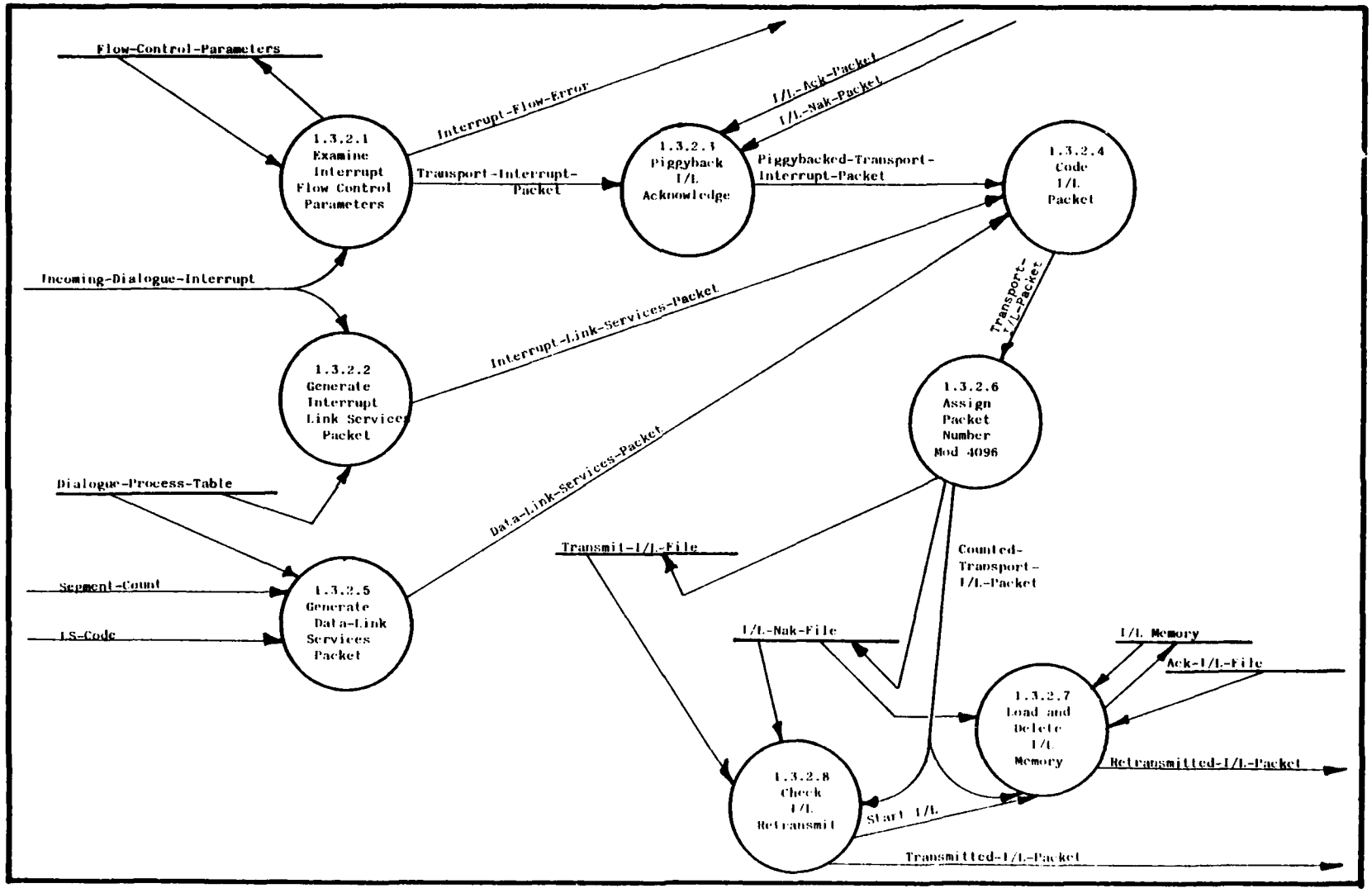


Figure C-18 Execute I/L Packet (1.3.2) DFD

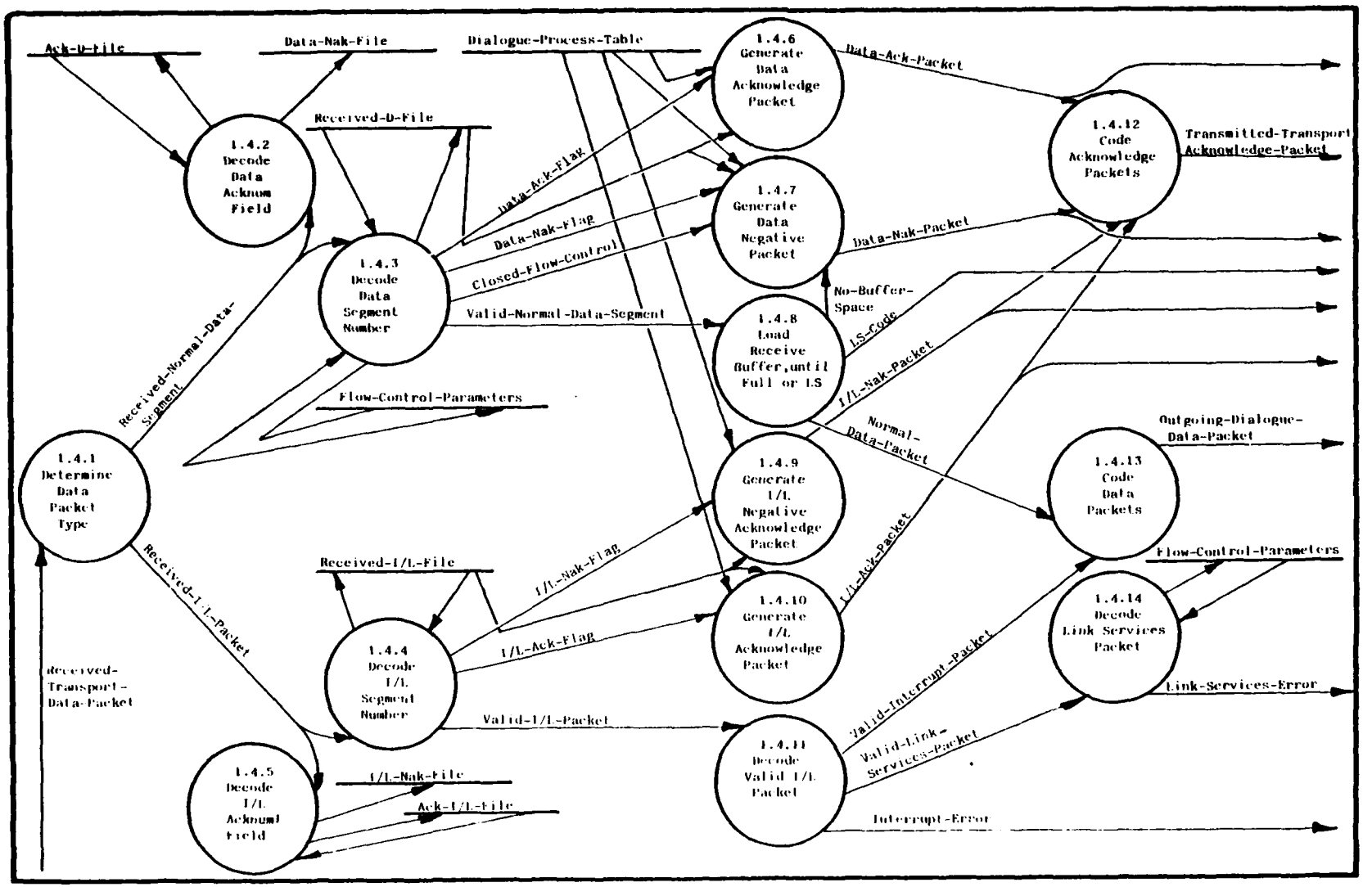


Figure C-19 Execute Data Packet (1.4) DFD

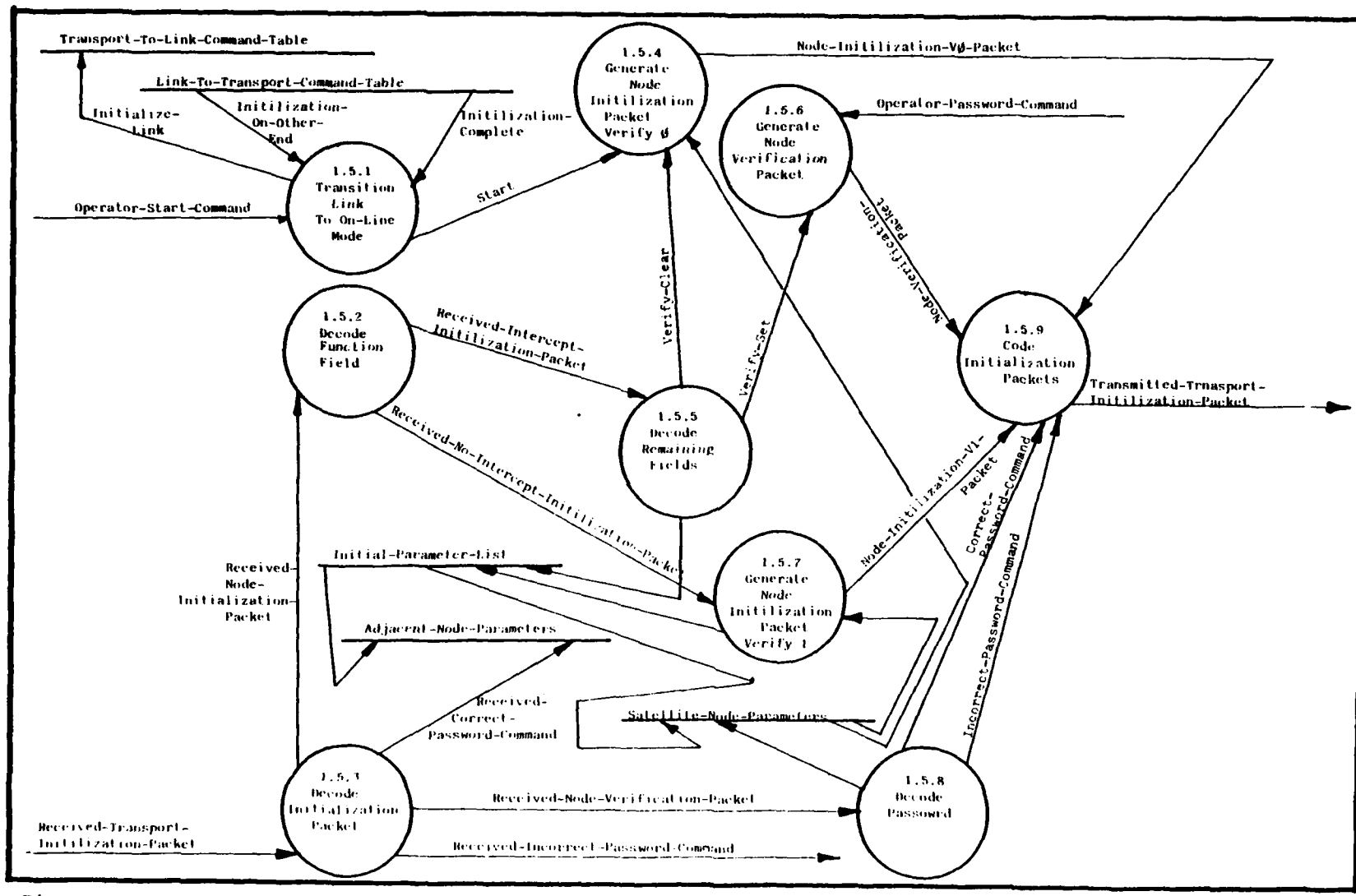


Figure C-20 Execute Startup Packet (1.5) DFD

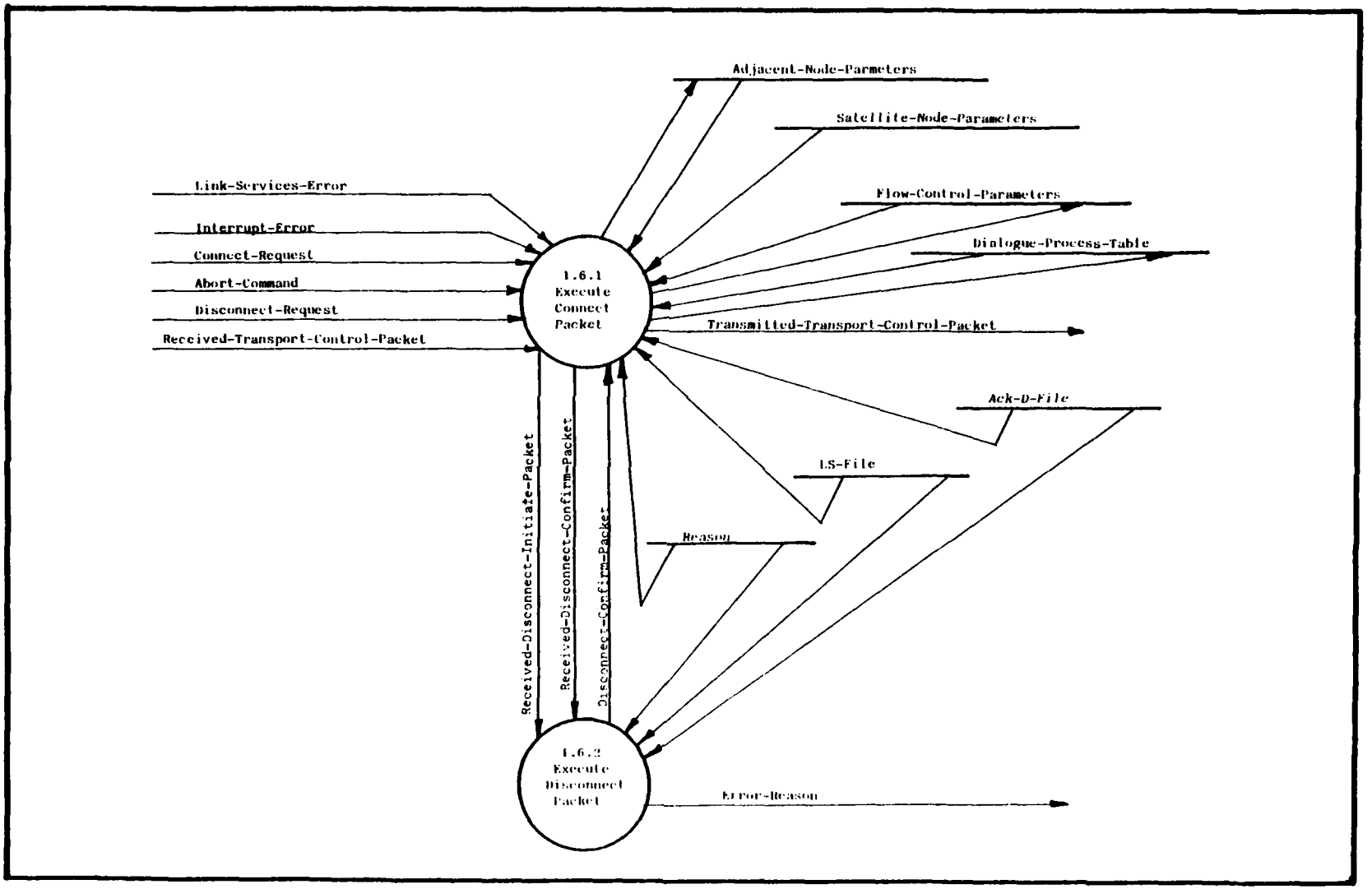


Figure C-21 Execute Control Packet (1.6) DFD

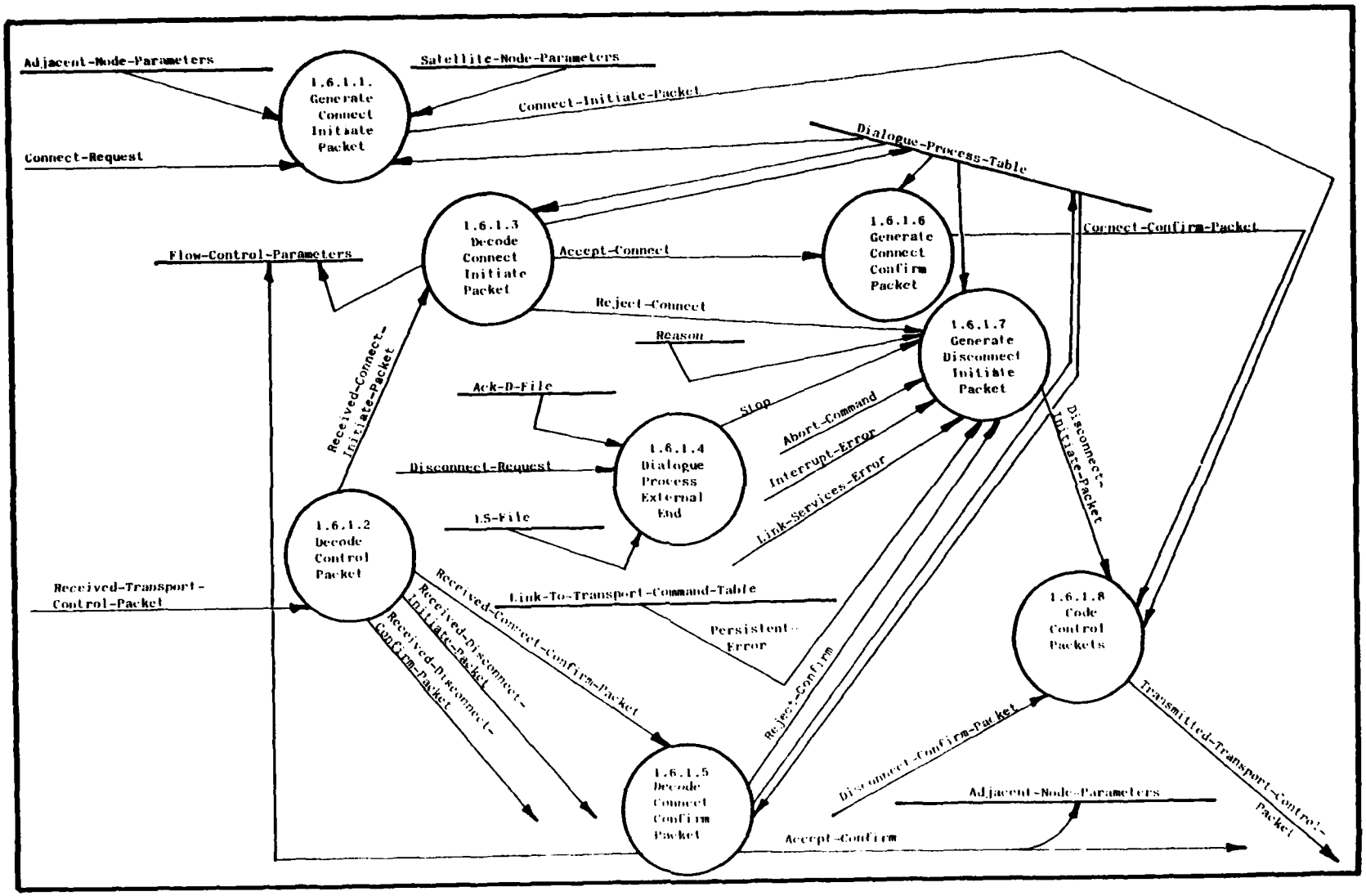


Figure C-22 Execute Connect Packet (1.6.1) DFD

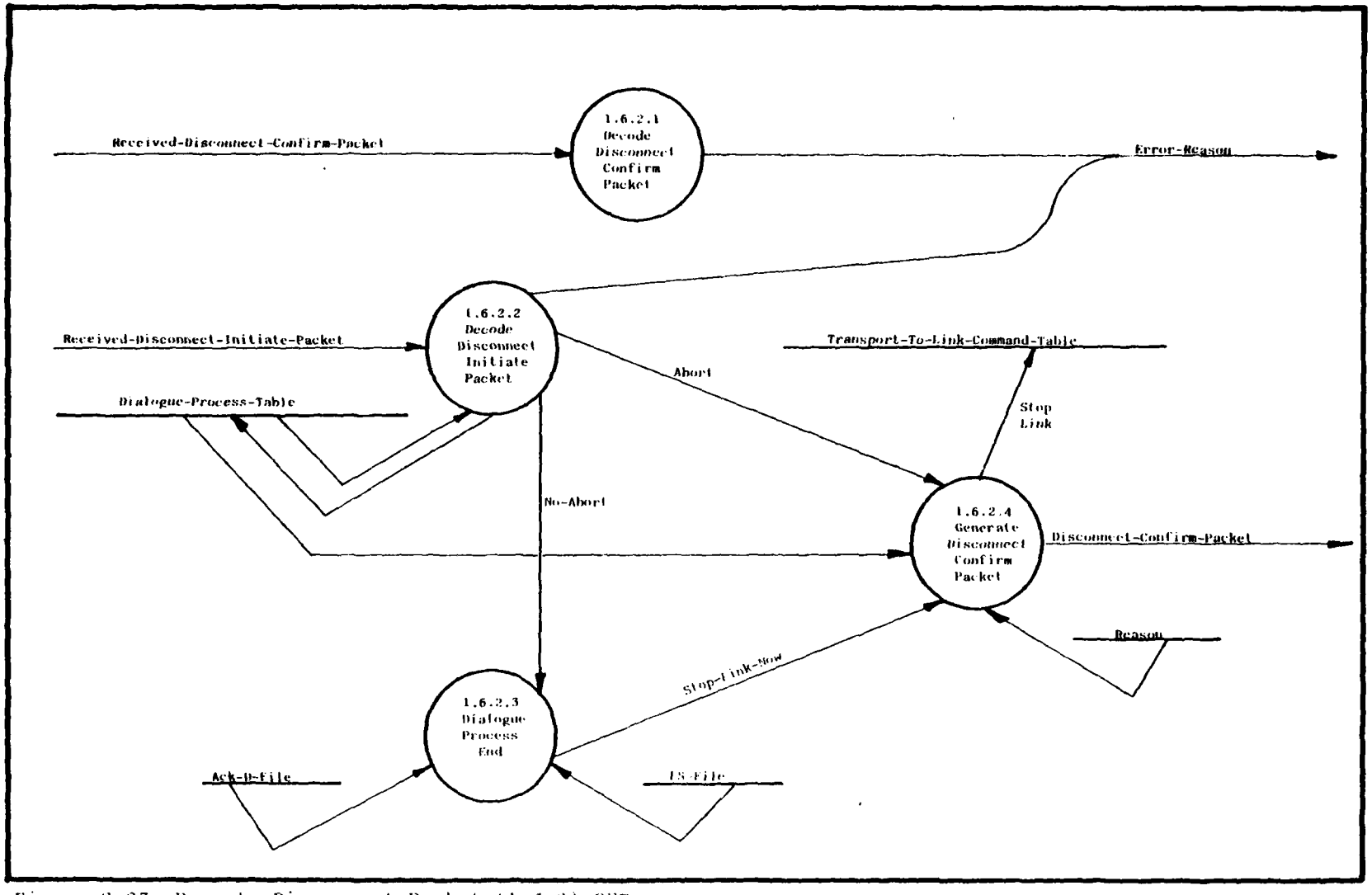


Figure C-23 Execute Disconnect Packet (1.6.2) DFD

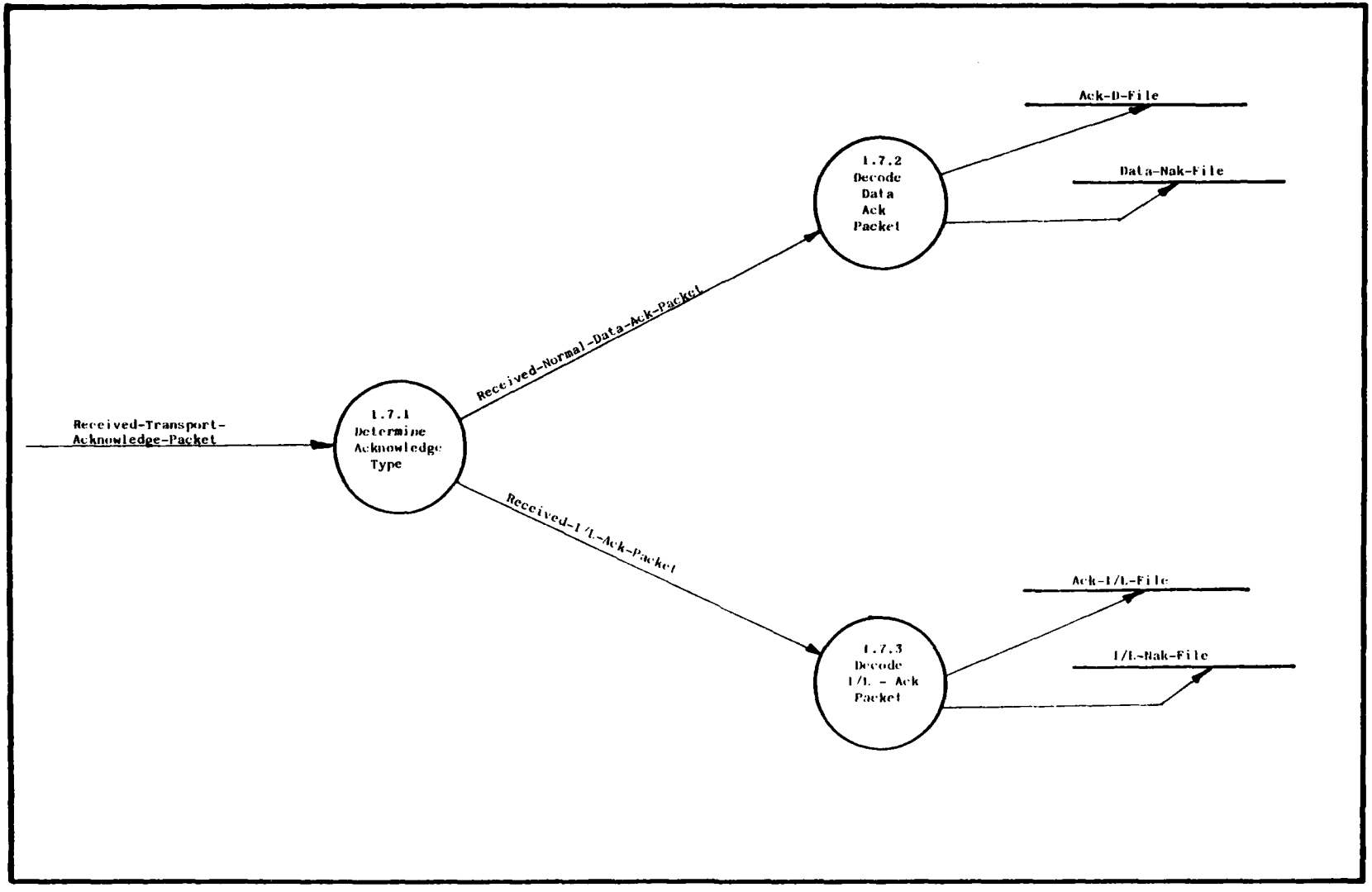


Figure C-24 Execute Acknowledge Packet (1.7) DFD

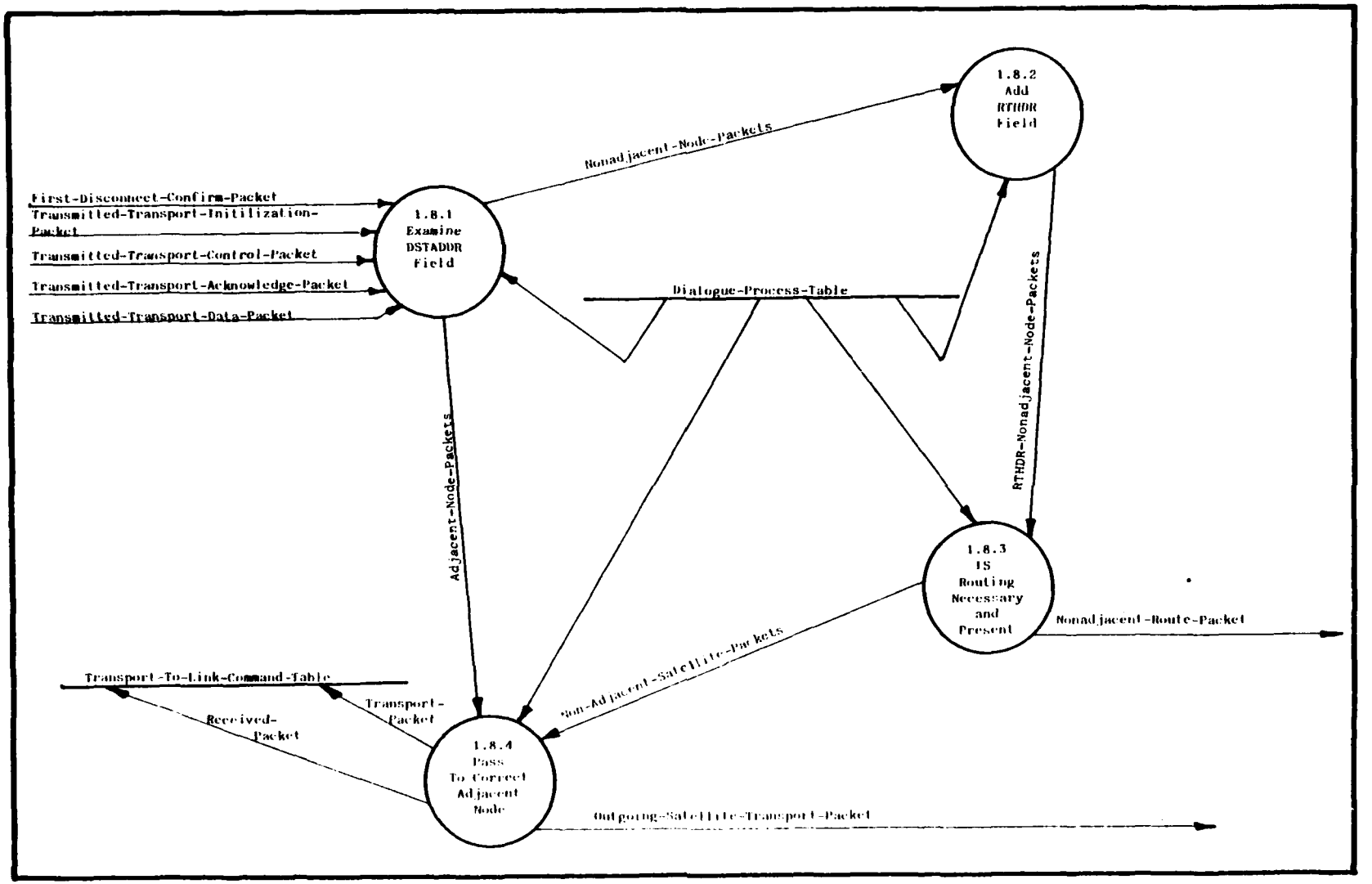


Figure C-25 Execute Outgoing Transport Packets (1.8) DFD

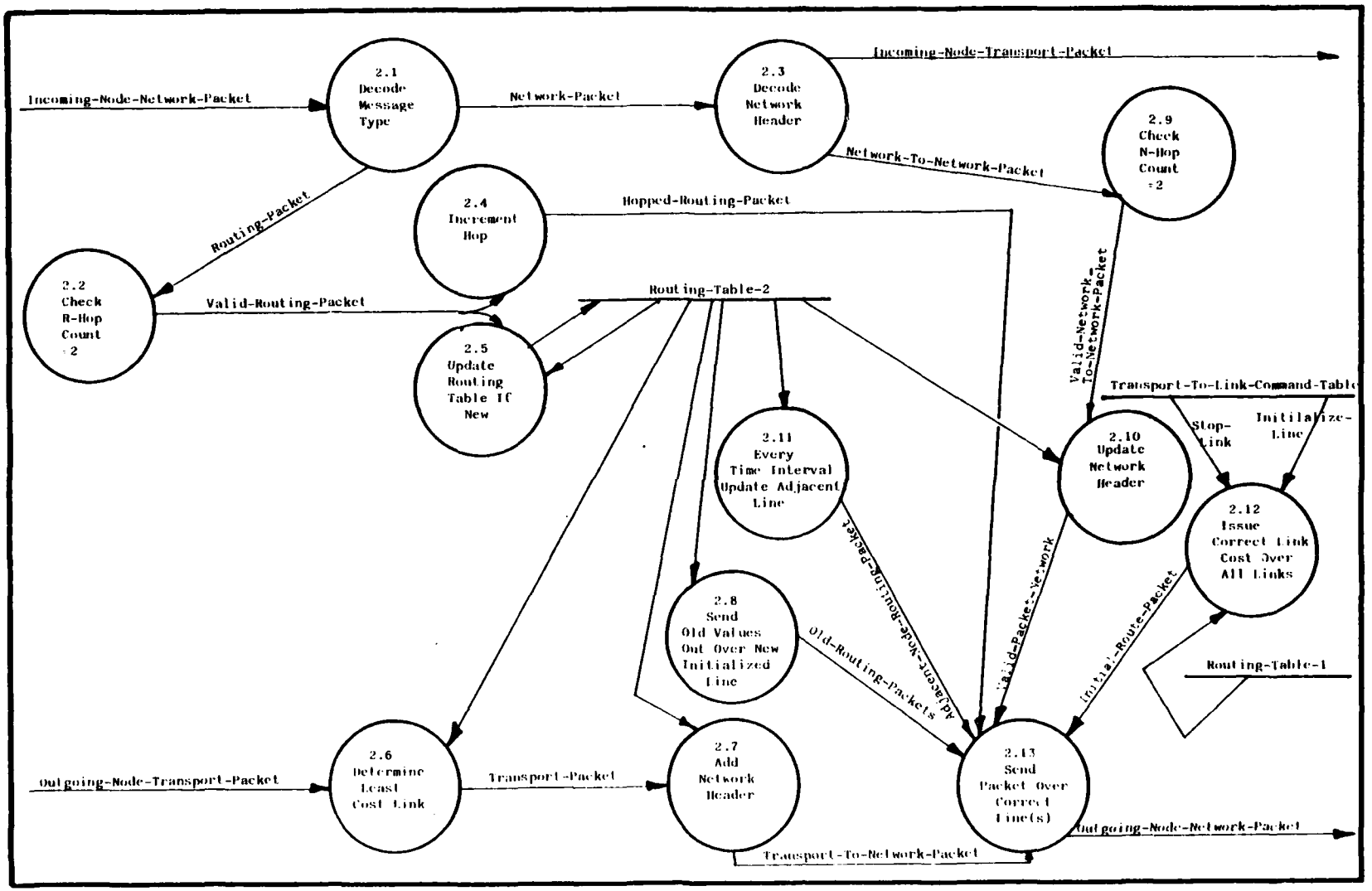


Figure C-26 Execute Network Protocol at Primary Node (2) DFD

Data Flow Diagrams for Data Link Level Protocol

Contents

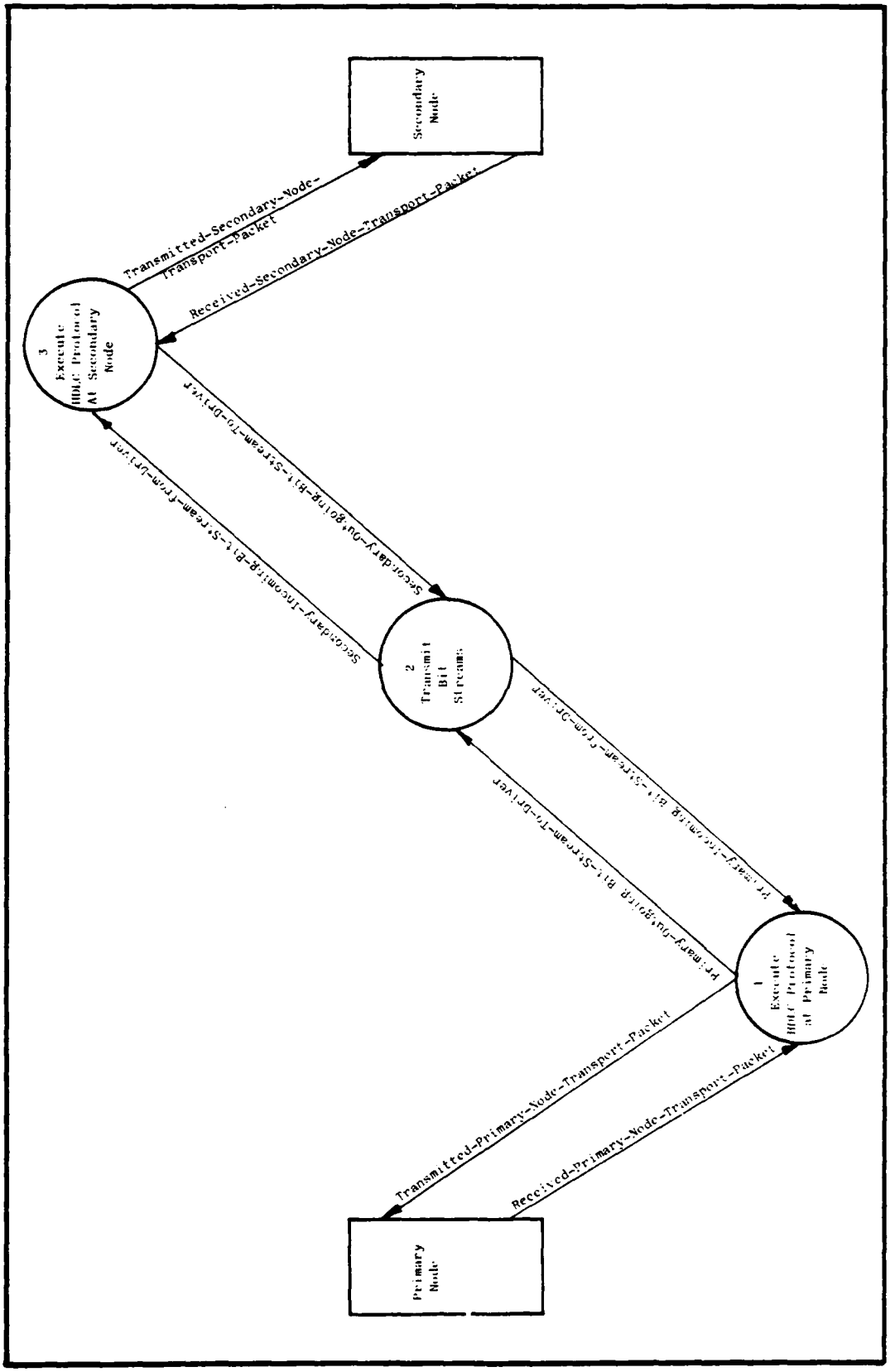


Figure C-27 Ewnet (HDLC) Overview DFD

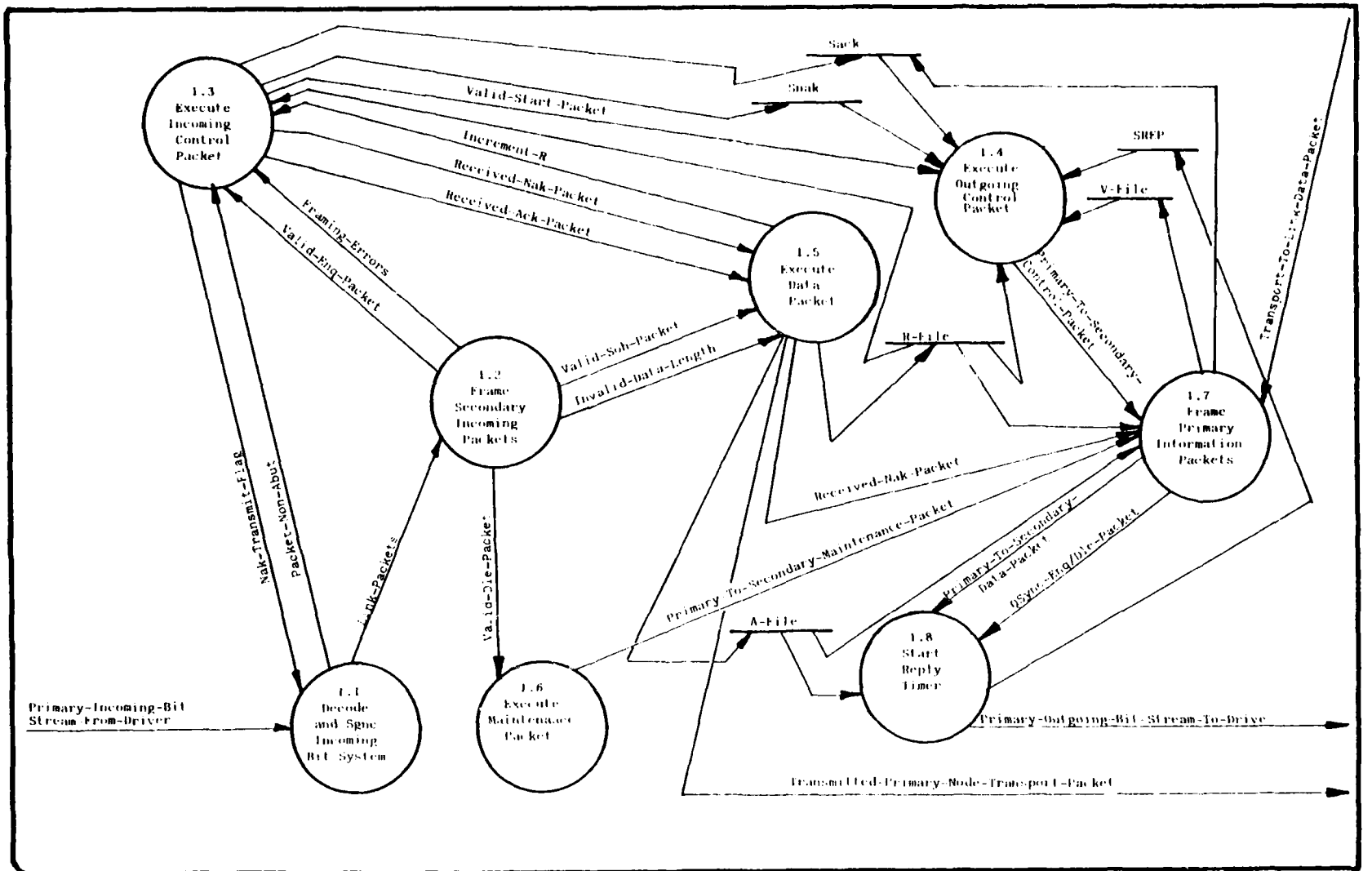


Figure C-28 Execute HDLC Protocol at Primary Node (1) DFD

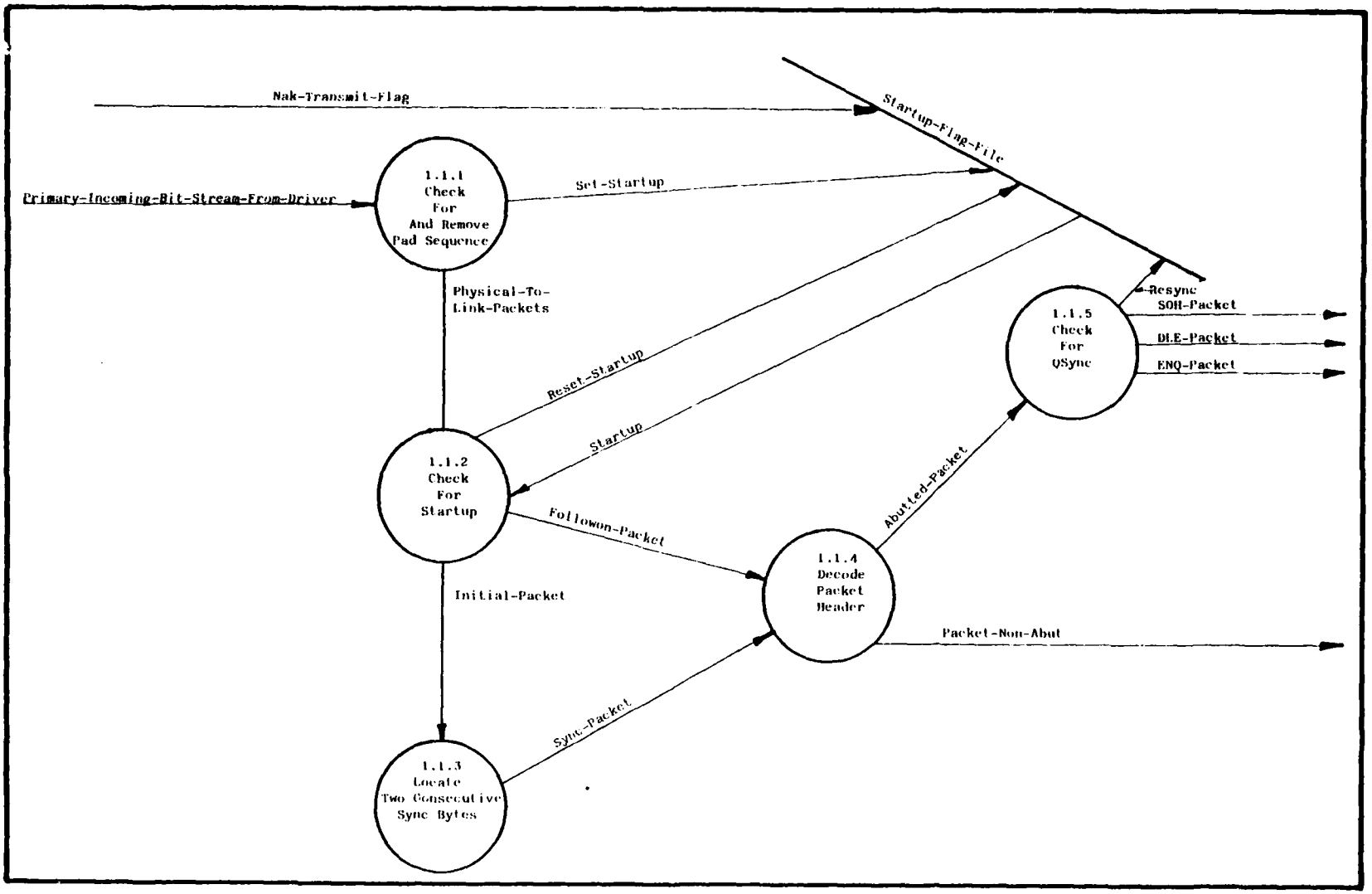


Figure C-29 Decode and Sync Incoming Bit Stream (1.1) DFD

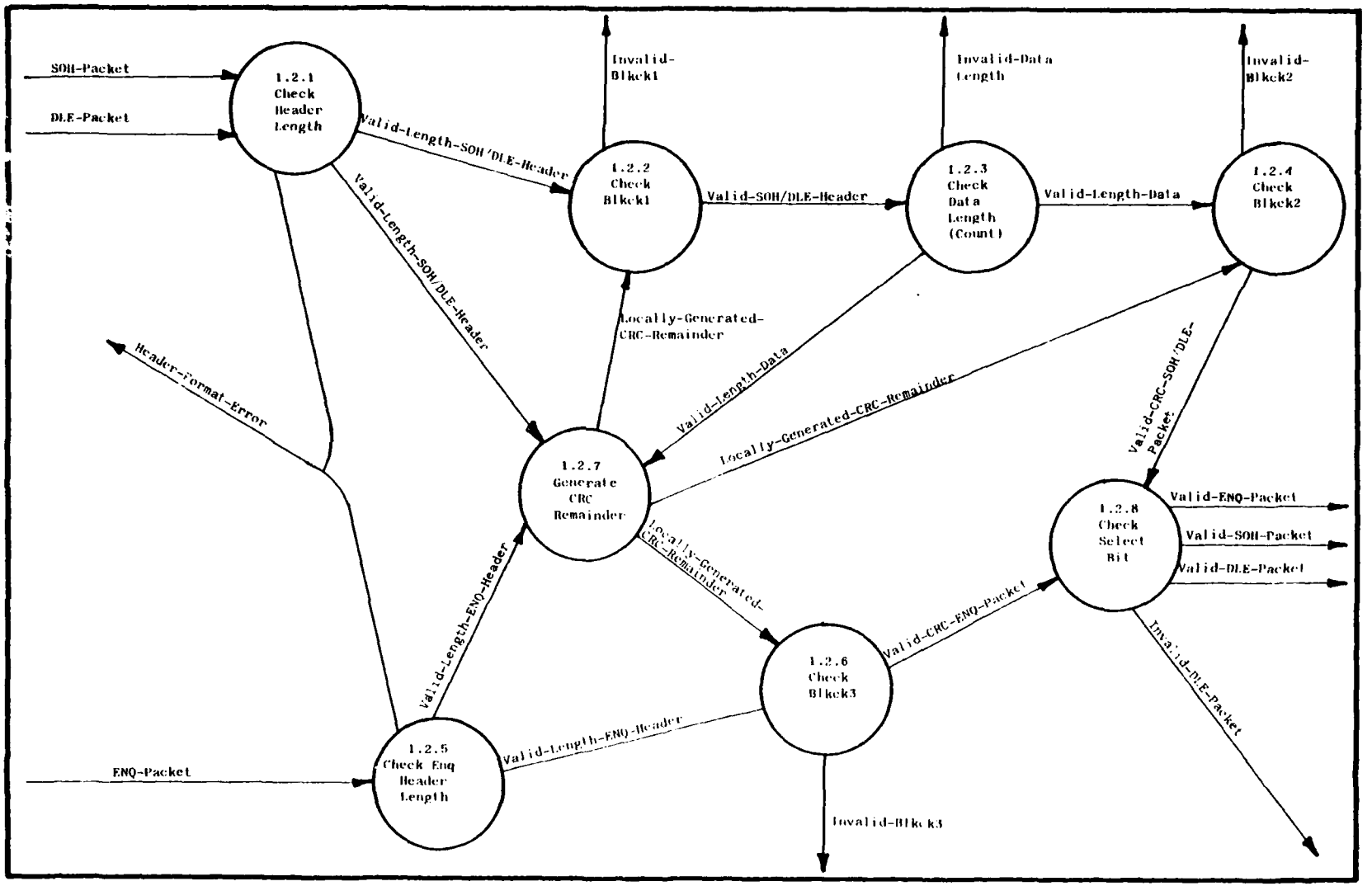


Figure C-30 Frame Secondary Information Packets (1.2) DFD

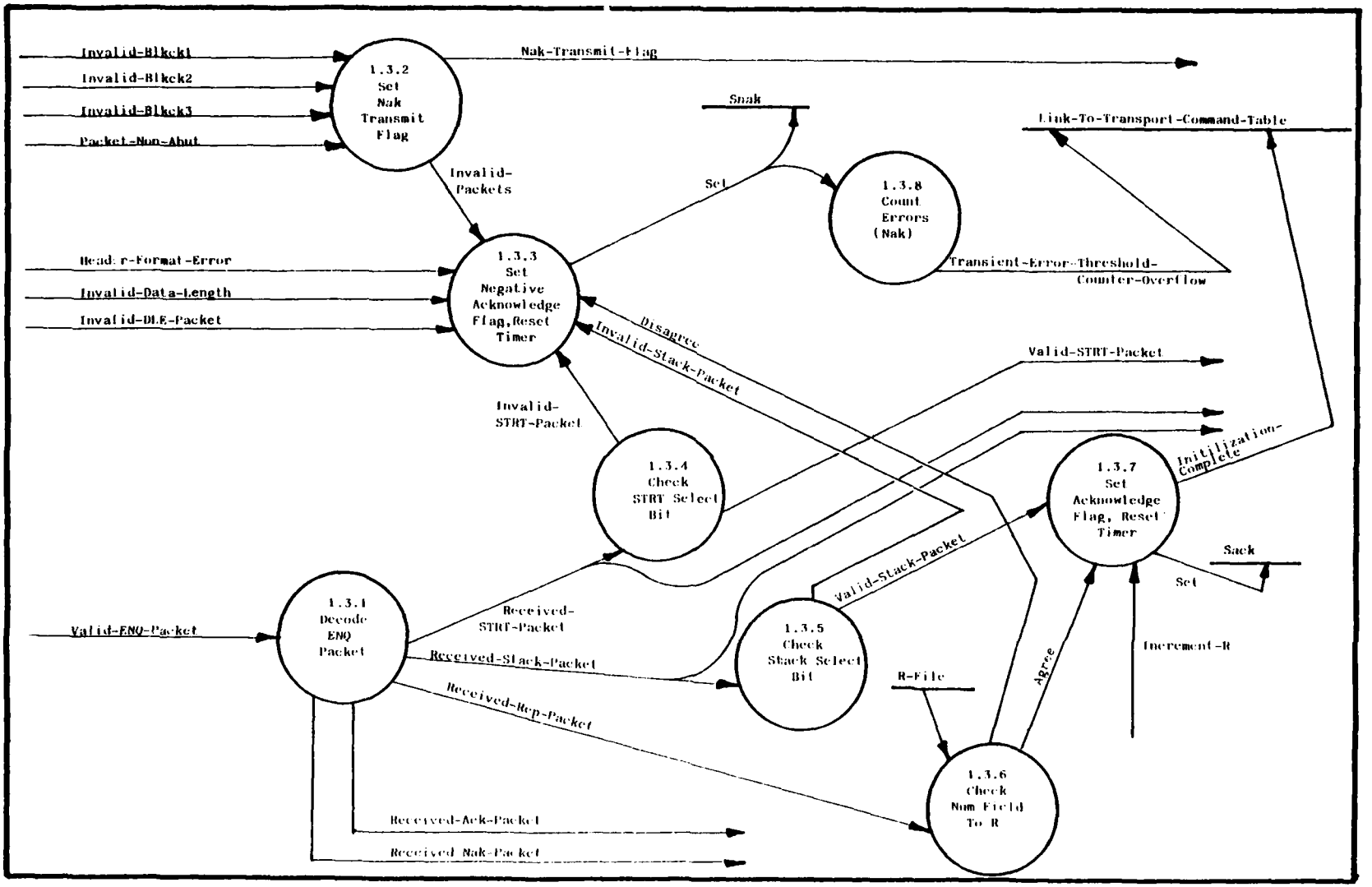


Figure C-31 Execute Incoming Control Packet (1.3) DFD

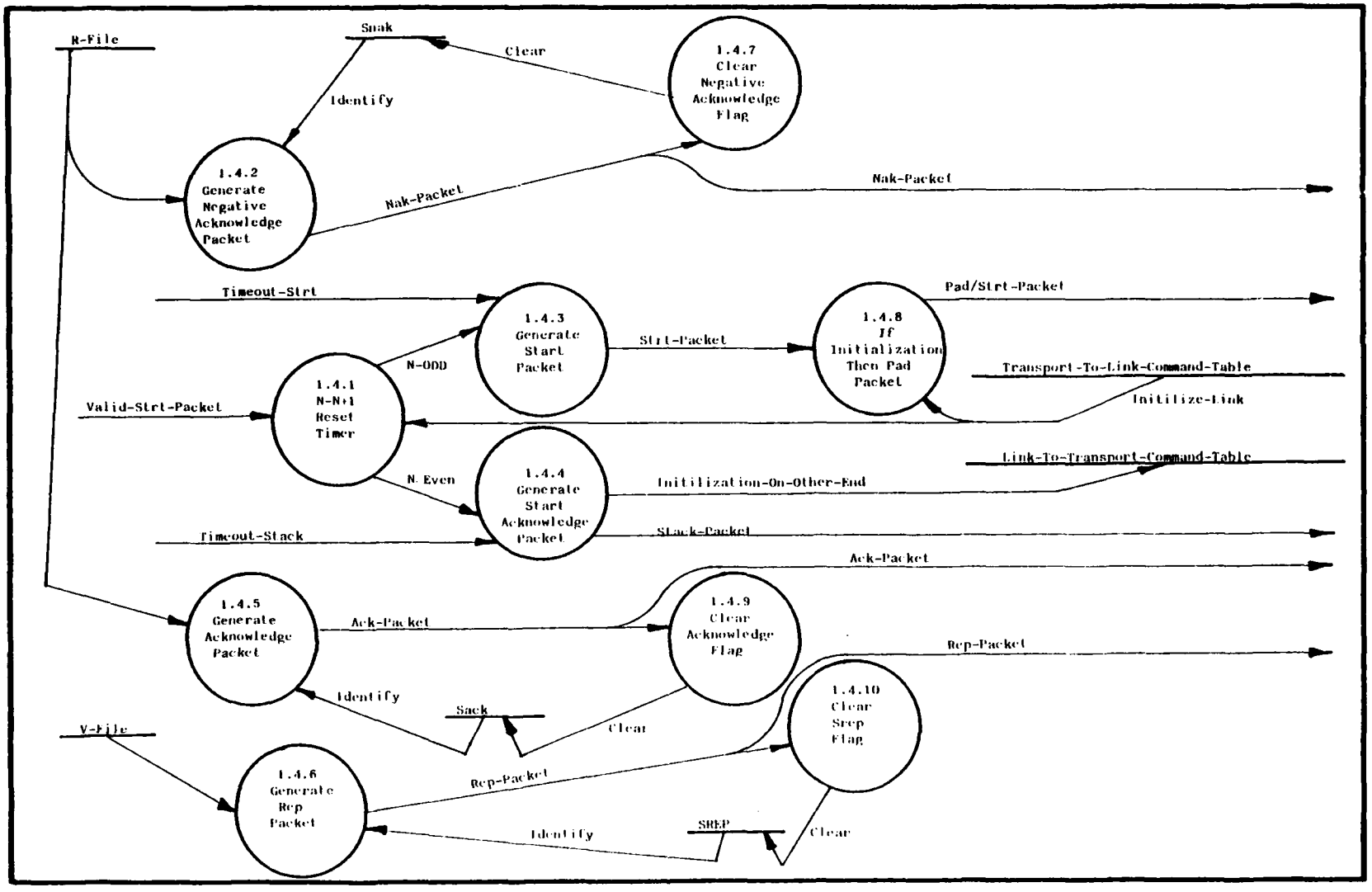


Figure C-32 Execute Outgoing Control Packet (1.4) DFD

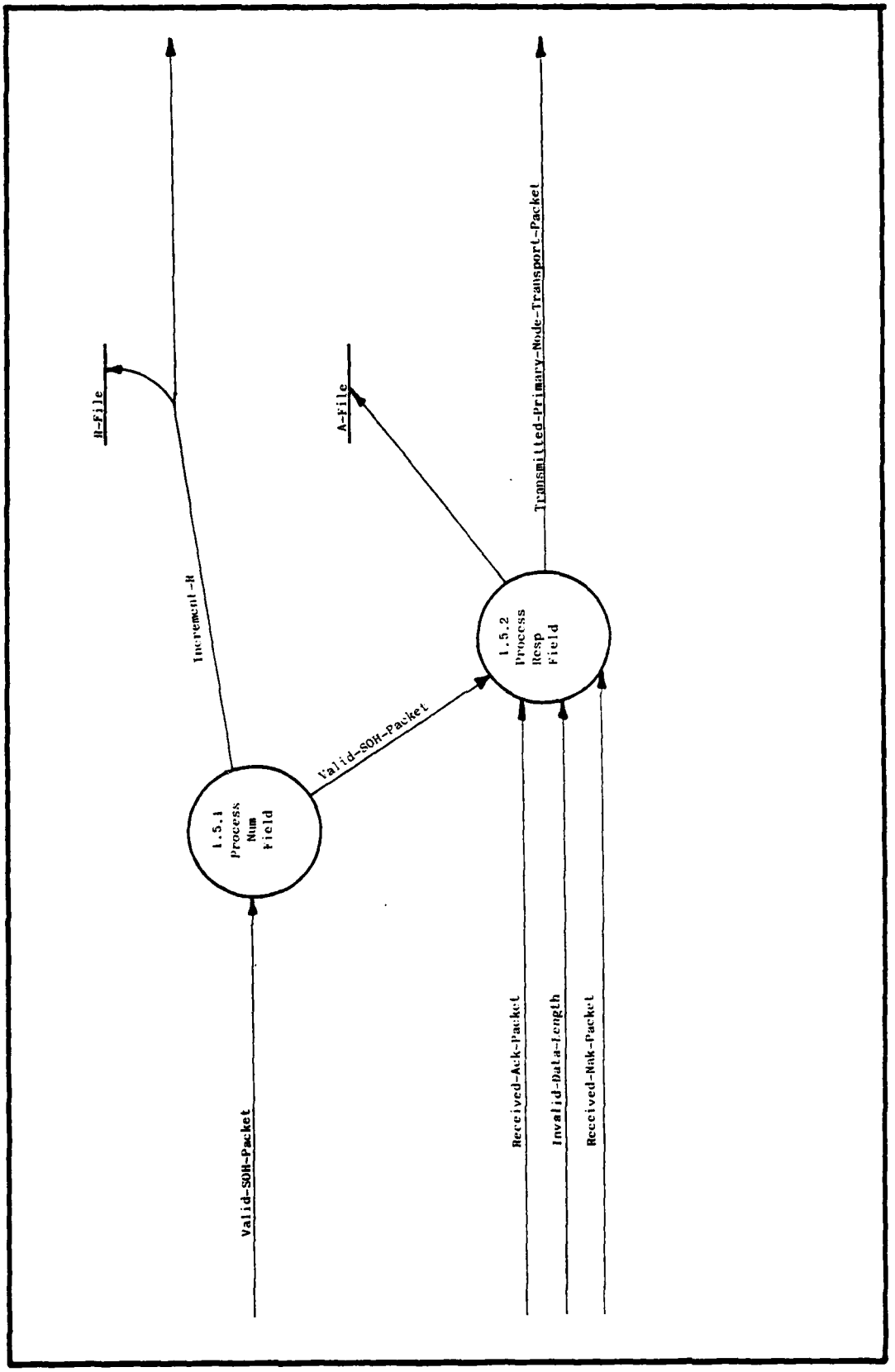


Figure C-33 Execute Data Packet (1.5) DFD

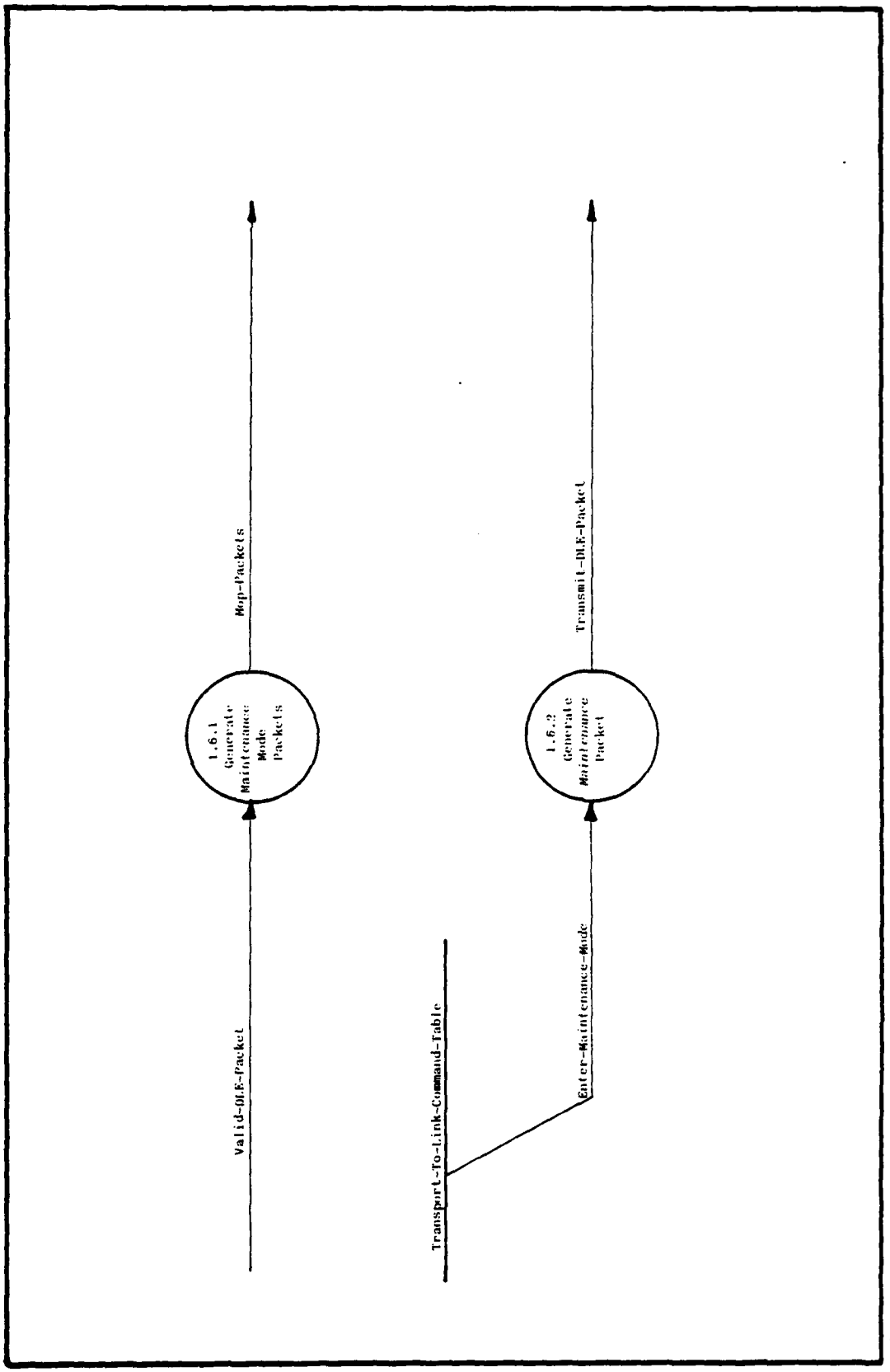


Figure C-34 Execute Maintenance Packet (1.6) DFD

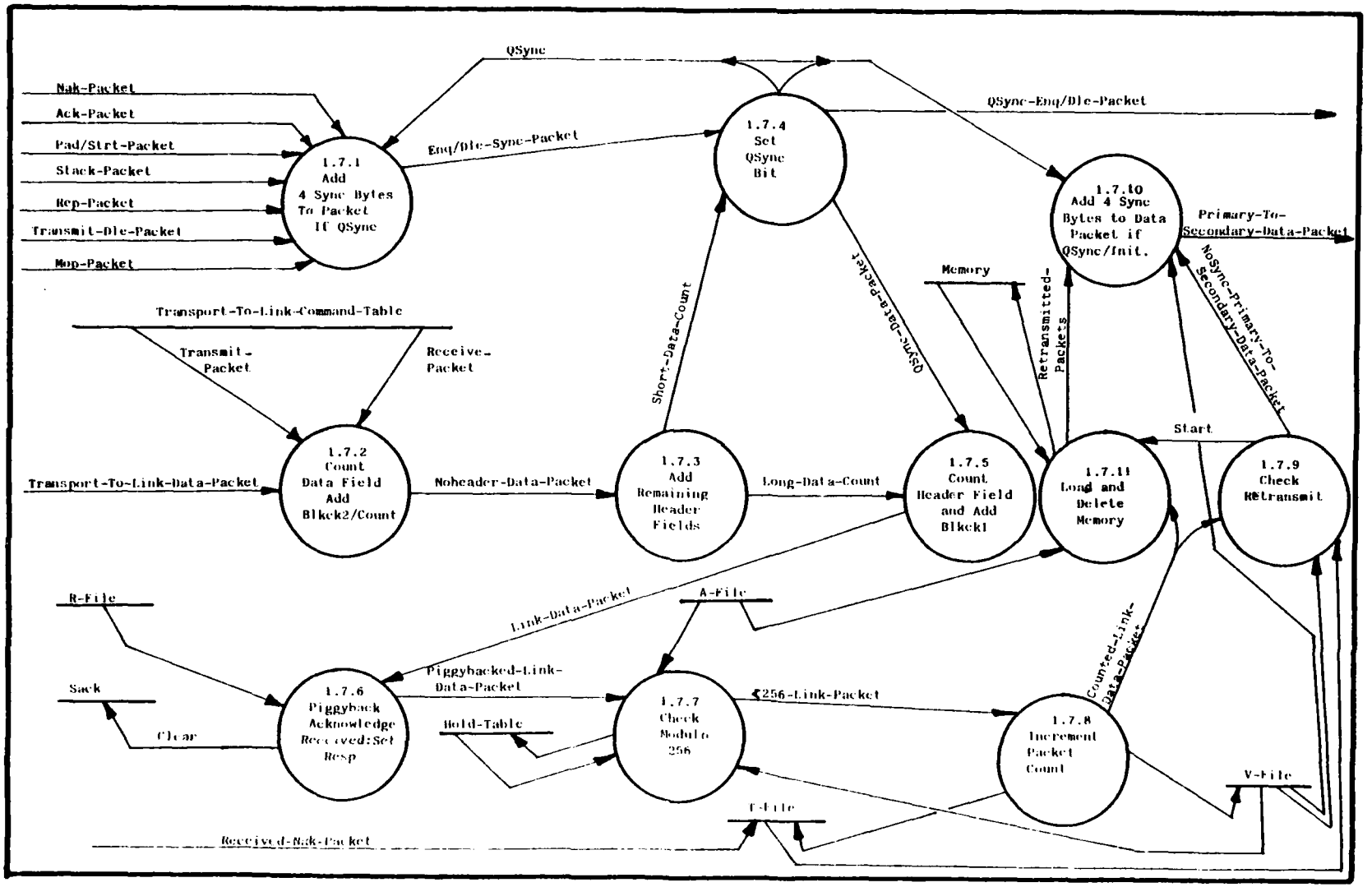


Figure C-35 Frame Primary Information Packets (1.7) DFD

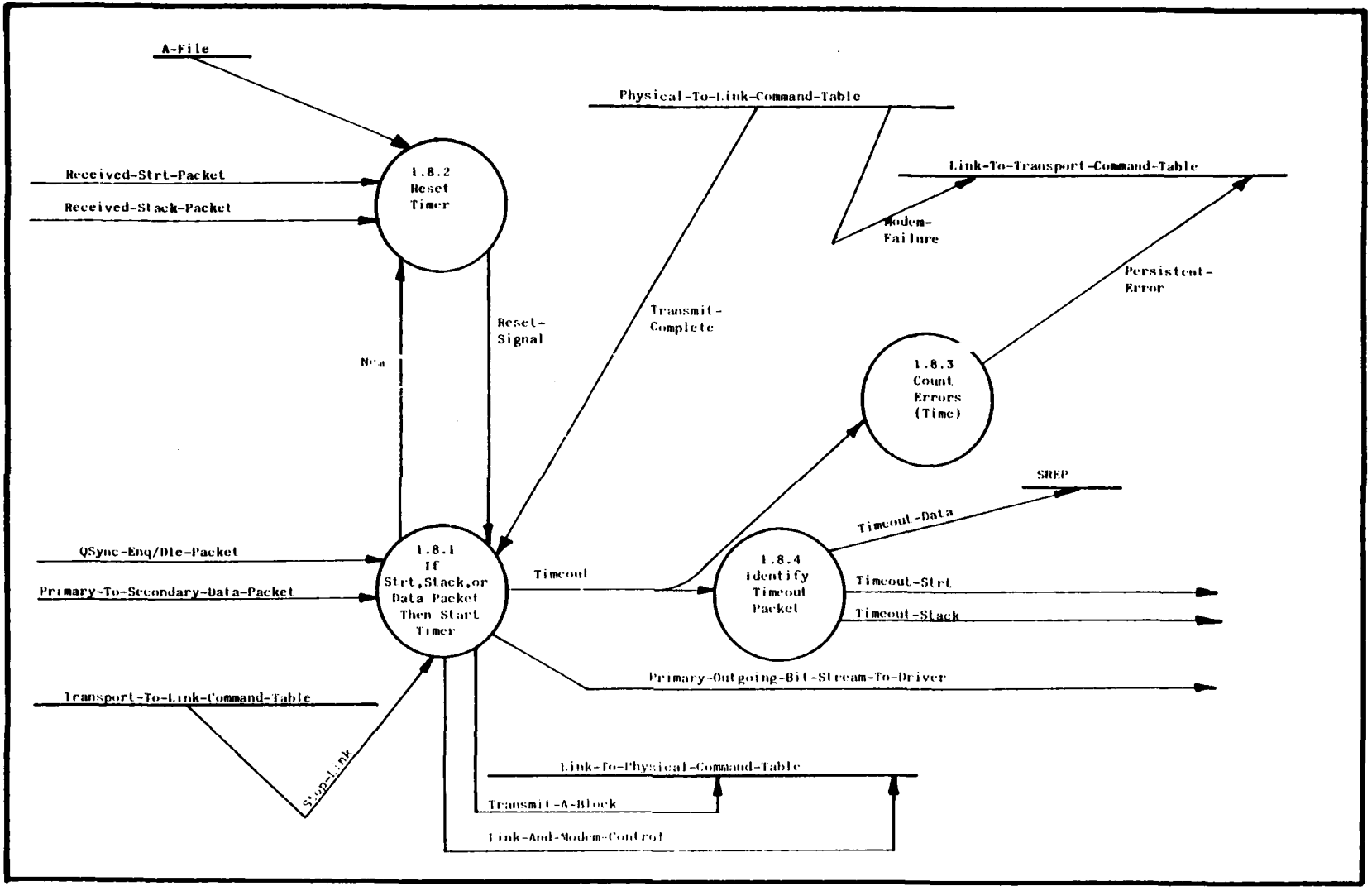


Figure C-36 Start Reply Timer (1.8) DFD

DATA DICTIONARY

FOR USER LEVEL (U) PROTOCOL

Contents

DATA ELEMENT NAME: ABORT_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL LAYER

DATA FLOW NAME: ARCHITECTURE_OPERATION_COMMANDS
ALIASES: NONE
COMPOSITION:
ARCHITECTURE_OPERATION_COMMANDS = PRIMARY_START_FILE_CONFIGURATION
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: CONNECT_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL

DATA ELEMENT NAME: DISCONNECT_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL LAYER

DATA FLOW NAME: DLE_PACKET
ALIASES: NONE
COMPOSITION:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL,
NOTES: EXECUTE LINK LEVEL PROTOCOL LAYER

DATA FLOW NAME: ENQ_PACKET
ALIASES: NONE
COMPOSTION:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE LINK LEVEL PROTOCOL LAYER

DATA FLOW NAME: ERROR_REASON
ALIASES: NONE
COMPOSITION:

REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.

NOTES: EXECUTE TRANSPORT PROTOCOL LAYER

DATA ELEMENT NAME: FILE_TRANSFER_INFORMATION
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO OBTAIN PROCEDURES FOR TRANSFERRING FILES.

NOTES: EXECUTE HELP COMMANDS LAYER

DATA FLOW NAME: FILE_TRANSFER_PROCEDURE
ALIASES: NONE
COMPOSITION:

FILE_TRANSFER_PROCEDURE = TABLE OF NECESSARY ACTIONS AND COMMANDS DEPENDING ON FILE_TRANSFER_TYPE (SEQUENTIAL, KEY, OR DATA FILE ADDRESS).

NOTES: EXECUTE HELP COMMANDS LAYER

DATA FLOW NAME: FLOW_CONTROL_ERRORS
ALIASES: NONE
COMPOSITION:

REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.

NOTES: EXECUTE TRANSPORT PROTOCOL LAYER

DATA ELEMENT NAME: GENERAL_INFORMATION
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO OBTAIN GENERAL NETWORK INFORMATION.

NOTES: EXECUTE HELP COMMANDS LAYER

DATA FLOW NAME: HELP_RESPONSE
ALIASES: NONE
COMPOSITION:

HELP_RESPONSE = [FILE_TRANSFER_PROCEDURE | MENU_SELECTION]

```
                                        | NETWORK_CONFIGURATION   |
                                        | STARTUP_PROCEDURE       |
                                        --                        --
```

NOTES: EXECUTE HELP COMMANDS LAYER
EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: INCOMING_ARCHITECTURE_PACKETS
ALIASES: OUTGOING_ARCHITECTURE_PACKETS
COMPOSITION:
INCOMING_ARCHITECTURE_PACKETS = INCOMING_PRIMARY_NODE_ARCHITECTURE_PACKETS
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: INCOMING_LINK_PACKETS
ALIASES: PRIMARY_IMCOMING_PHYSICAL_PACKETS
IMCOMING_PHYSICAL_PACKETS
OUTGOING_PHYSICAL_PACKETS
OUTGOING_LINK_PACKETS
SECONDARY_INCOMING_PHYSICAL_PACKETS
COMPOSITION:
INCOMING_LINK_PACKETS = (11111111) + (2{SYNC}8) +

```
                                                          __          __
                                                        | SOH_PACKET  |
                                                        | DLE_PACKET  |
                                                        | ENQ_PACKET  |
                                                        --          --
```

NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: INCOMING_NODE_NETWORK_PACKET
ALIASES: NONE
COMPOSITION:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE NETWORK LEVEL PROTOCOL

DATA FLOW NAME: INCOMING_PHYSICAL_PACKETS
ALIASES: INCOMING_LINK_PACKETS
OUTGOING_PHYSICAL_PACKETS
PRIMARY_INCOMING_PHYSICAL_PACKETS
OUTGOING_LINK_PACKETS
SECONDARY_INCOMING_PHYSICAL_PACKETS
COMPOSITION:
SEE ALIASES

NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: INCOMING_PRIMARY_NODE_ARCHITECTURE_PACKETS
ALIASES: NONE
COMPOSITION:

REFER TO DATA DICTIONARY FOR ARCHITECTURE LEVEL PROTOCOL.
NOTES: EXECUTE ARCHITECTURE LEVEL PROTOCOL

DATA FLOW NAME: INCOMING_SATELLITE_TRANSPORT_PACKET
ALIASES: NONE
COMPOSITION:

REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL

DATA FLOW NAME: INCOMING_TRANSPORT_PACKETS
ALIASES: OUTGOING_TRANSPORT_PACKETS
COMPOSITION:

INCOMING_TRANSPORT_PACKETS = INCOMING_SATELLITE_TRANSPORT_PACKET
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: LIST_OF_ACTIVE_HOST_AND_DEVICE_NAMES
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO OBTAIN THE CURRENT NETWORK CONFIGURATION.
NOTES: EXECUTE HELP COMMANDS LAYER

DATA FLOW NAME: LOCAL_COMMAND
ALIASES: NONE
COMPOSITION:

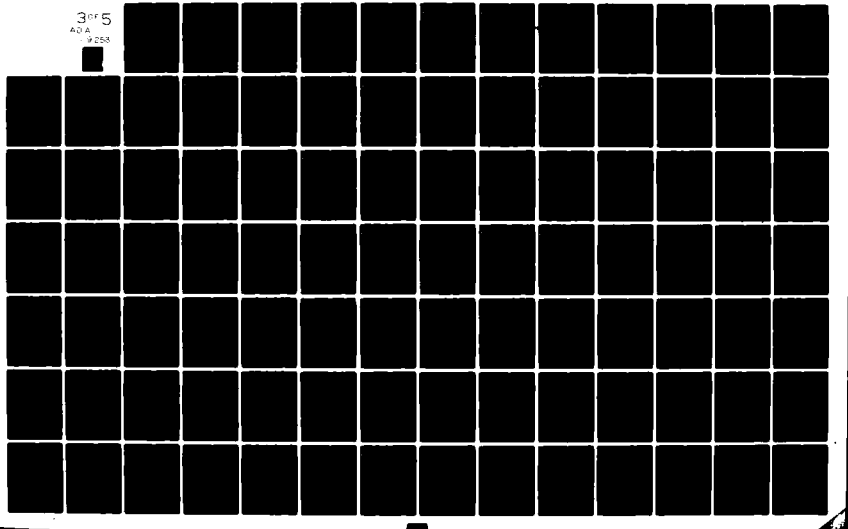
LOCAL_COMMAND = ALL LOCAL COMPUTER COMMANDS.
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: MENU_SELECTION
ALIASES: NONE
COMPOSITION:

MENU_SELECTION = $\overline{|}$ LIST_OF_ACTIVE_HOST_AND_DEVICE_NAMES $\overline{|}$

AD-A119 253 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONI--ETC(U)
DEC 81 R H STOKES

UNCLASSIFIED AFIT/GCS/EE/81D-16 NL



```
                                        | FILE_TRANSFER_INFORMATION          |
                                        | PROTOCOL_STARTUP_INFORMATION       |
                                        --                                   --
NOTES:                  EXECUTE HELP COMMANDS LAYER


DATA FLOW NAME:         NETWORK_CONFIGURATION
ALIASES:                NONE
COMPOSITION:
                        NETWORK_CONFIGURATION = ALL DATA IN THE DIALOGUE_PROCESS_
                                                TABLE.
NOTES:                  EXECUTE HELP COMMANDS LAYER


DATA FLOW NAME:         NETWORK_HELP_COMMAND
ALIASES:                NONE
COMPOSITION:
                        NETWORK_HELP_COMMAND =

                                       __                                    __
                                      | LIST_OF_ACTIVE_HOST_AND_DEVICE_NAMES |
                                      | FILE_TRANSFER_INFORMATION            |
                                      | GENERAL_INFORMATION                  |
                                      | PROTOCOL_START_INFORMATION           |
                                      --                                     --
NOTES:                  EXECUTE HELP COMMANDS LAYER
                        EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER


DATA FLOW NAME:         OPERATOR_COMMANDS
ALIASES:                PRIMARY_OPERATOR_COMMANDS
                        SECONDARY_OPERATOR_COMMANDS
COMPOSITION:
                                                  __                          __
                        OPERATOR_COMMANDS = | ARCHITECTURE_OPERATION_COMMANDS |
                                            | TRANSPORT_OPERATION_COMMANDS    |
                                            | LOCAL_COMMAND                   |
                                            | NETWORK_HELP_COMMANDS           |
                                            --                                --
NOTES:                  EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER


DATA FLOW NAME:         OPERATOR_DISPLAY
ALIASES:                PRIMARY_OPERATOR_DISPLAY
                        SECONDARY_OPERATOR_DISPLAY
COMPOSITION:
                        SEE ALIASES
NOTES:                  EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER
```

DATA ELEMENT NAME: OPERATOR_PASSWORD_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL LAYER

DATA ELEMENT NAME: OPERATOR_START_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT LEVEL PROTOCOL LAYER

DATA FLOW NAME: OUTGOING_ARCHITECTURE_PACKETS
ALIASES: INCOMING_ARCHITECTURE_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_LINK_PACKETS
ALIASES: INCOMING_LINK_PACKETS
PRIMARY_INCOMING_PHYSICAL_PACKETS
INCOMING_PHYSICAL_PACKETS
OUTGOING_PHYSICAL_PACKETS
SECONDARY_INCOMING_PHYSICAL_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_NETWORK_PACKETS
ALIASES: INCOMING_NETWORK_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_PHYSICAL_PACKETS
ALIASES: INCOMING_LINK_PACKETS
PRIMARY_INCOMING_PHYSICAL_PACKETS

INCOMING_PHYSICAL_PACKETS
OUTGOING_LINK_PACKETS
SECONDARY_INCOMING_PHYSICAL_PACKETS

COMPOSITION: SEE ALIASES

NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_TRANSPORT_PACKETS
ALIASES: INCOMING_TRANSPORT_PACKETS
COMPOSITION: SEE ALIASES
NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: PRIMARY_INCOMING_PHYSICAL_PACKETS
ALIASES: SECONDARY_INCOMING_PHYSICAL_PACKETS
INCOMING_PHYSICAL_PACKETS
OUTGOING_PHYSICAL_PACKETS
INCOMING_LINK_PACKETS
OUTGOING_LINK_PACKETS
COMPOSITION: SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: PRIMARY_OPERATOR_COMMANDS
ALIASES: OPERATOR_COMMANDS
SECONDARY_OPERATOR_COMMANDS
COMPOSITION: SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: PRIMARY_OPERATOR_DISPLAY
ALIASES: OPERATOR_DISPLAY
SECONDARY_OPERATOR_DISPLAY
COMPOSITION:

```
                                  __                    __
PRIMARY_OPERATOR_DISPLAY = |  TRANSPORT_DISPLAY_COMMANDS  |
                           |  HELP_RESPONSE               |
                           --                            --
```

NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: PRIMARY_START_FILE_CONFIGURATION
ALIASES: NONE

VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR ARCHITECTURE LEVEL PROTOCOL.

NOTES: EXECUTE ARICHITECTURE LEVEL PROTOCOL LAYER

DATA ELEMENT NAME: PROTOCOL_START_INFORMATION
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO OBTAIN STARTUP PROCEDURES.

NOTES: EXECUTE HELP COMMANDS LAYER

DATA ELEMENT NAME: RECEIVED_INCORRECT_PASSWORD_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.

NOTES: EXECUTE TRANSPORT PROTOCOL LAYER

DATA FLOW NAME: SECONDARY_INCOMING_PHYSICAL_PACKETS
ALIASES: INCOMING_LINK_PACKETS
PRIMARY_INCOMING_PHYSICAL_PACKETS
INCOMING_PHYSICAL_PACKETS
OUTGOING_PHYSICAL_PACKETS
OUTGOING_LINK_PACKETS
COMPOSITION:
SEE ALIASES

NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OPERATOR_COMMANDS
ALIASES: OPERATOR_COMMANDS
PRIMARY_OPERATOR_COMMANDS
COMPOSITION:
SEE ALIASES

NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OPERATOR_DISPLAY
ALIASES: PRIMARY_OPERATOR_DISPLAY
OPERATOR_DISPLAY
COMPOSITION:
SEE ALIASES

NOTES: OVERVIEW LAYER

DATA FLOW NAME: SOH_PACKET
ALIASES: NONE
COMPOSITION:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE LINK LEVEL PROTOCOL LAYER

DATA FLOW NAME: STARTUP_PROCEDURES
ALIASES: NONE
COMPOSITION:
STARTUP_PROCEDURES = A TABLE THAT CONTAINS ALL COMMANDS
AND ACTIONS OF A NETWORK STARTUP.
NOTES: EXECUTE HELP COMMANDS LAYER

DATA FLOW NAME: SYNC
ALIASES: NONE
COMPOSITION:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE LINK LEVEL PROTOCOL LAYER

DATA FLOW NAME: TRANSPORT_DISPLAY_COMMANDS
ALIASES: NONE
COMPOSITION:
TRANSPORT_DISPLAY_COMMANDS =

```
                    __                                    --
                   | RECEIVED_INCORRECT_PASSWORD_COMMAND  |
                   | ERROR_REASON                         |
                   | FLOW_CONTROL_ERRORS                  |
                   --                                    --
```

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: TRANSPORT_OPERATION_COMMANDS
ALIASES: NONE
COMPOSITION:
TRANSPORT_OPERATION_COMMANDS =

```
                              __                          __
                             | OPERATOR_PASSWORD_COMMAND  |
                             | OPERATOR_START_COMMAND     |
                             | ABORT_COMMAND              |
                             | DISCONNECT_REQUEST         |
                             | CONNECT_REQUEST            |
                             --                          --
```

NOTES: EXECUTE EWNET PROTOCOL AT PRIMARY NODE LAYER

FILE DEFINITIONS (U)

FILE OR DATABASE NAME:	DIALOGUE_PROCESS_TABLE
ALIASES:	NONE
COMPOSITION:	
	REFER TO DATA DICTIONARY FOR THE TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES:	EXECUTE HELP COMMANDS LAYER

PROCESS SPECIFICATION (U)

```
PROCESS NAME:          DETERMINE COMMAND TYPE
PROCESS NUMBER:        1.1
PROCESS DESCRIPTION:
IF USER_COMMAND contains NETWORK_ID and COMMAND_FIELD = TRANSFER_COMMAND then
     Output command as ARCHITECTURE_OPERATION_COMMANDS
ELSEIF USER_COMMAND contains NETWORK_ID and COMMAND_FIELD = LOGICAL_LINK_
     ESTABLISHMENT then
          Output command as TRANSPORT_OPERATION_COMMANDS
ELSEIF USER_COMMAND contains NETWORK_ID and COMMAND_FIELD = HELP then
     Output command as NETWORK_HELP_COMMAND
ELSE output command as LOCAL_COMMAND


PROCESS NAME:          DECODE HELP COMMAND
PROCESS NUMBER:        1.2.1
PROCESS DESCRIPTION:
IF NETWORK_HELP_COMMAND contains HELP_FIELD = General information then
     Output GENERAL_INFORMATION Flag
ELSEIF NETWORK_HELP_COMMAND contains HELP_FIELD = File transfer then
     Output FILE_TRANSFER_INFORMATION Flag
ELSEIF NETWORK_HELP_COMMAND contains HELP_FIELD = Start then
     Output PROTOCOL_START_INFORMATION
ELSE output LIST_OF_ACTIVE_HOST_AND_DEVICE_NAMES Flag


PROCESS NAME:          PROVIDE GENERAL NETWORK INFO
PROCESS NUMBER:        1.2.2
PROCESS DESCRIPTION:
IF Input = GENERAL_INFORMATION Flag then
     Output MENU_SELECTION
ELSE Null


PROCESS NAME:          PROVIDE PROCEDURE FOR TRANSFERING FILES
PROCESS NUMBER:        1.2.3
PROCESS DESCRIPTION:
IF Input = FILE_TRANSFER_INFORMATION Flag then
     Output FILE_TRANSFER_PROCEDURE
ELSE Null


PROCESS NAME:          PROVIDE LIST OF ACTIVE HOSTS AND DEVICES
```

```
PROCESS NUMBER:       1.2.4
PROCESS DESCRIPTION:
IF Input = LIST_OF_ACTIVE_HOST_AND_DEVICE_NAMES Flag then
     Extract from DIALOGUE_PROCESS_TABLE and
          Output NETWORK_CONFIGURATION
ELSE Null


PROCESS NAME:         PROVIDE NETWORK STARTUP PROCEDURES
PROCESS NUMBER:       1.2.5
PROCESS DESCRIPTION:
IF Input = PROTOCOL_START_INFORMATION Flag then
     Output STARTUP_PROCEDURES
ELSE Null


PROCESS NAME:         CODE HELP RESPONSES
PROCESS NUMBER:       1.2.6
PROCESS DESCRIPTION:
IF Input = MENU_SELECTION, FILE_TRANSFER_PROCEDURES, NETWORK_CONFIGURATION,
     or STARTUP_PROCEDURES then
          Output as HELP_RESPONSE
ELSE Null


PROCESS NAME:         EXECUTE ARCHITECTURE LEVEL PROTOCOL
PROCESS NUMBER:       1.3
PROCESS DESCRIPTION:
EXECUTE the necessary processes to provide standardized formats and procedures
     for accessing and passing data between a user process and a file system
     existing in a network enviroment.


PROCESS NAME:         EXECUTE TRANSPORT LEVEL PROTOCOL
PROCESS NUMBER:       1.4
PROCESS DESCRIPTION:
EXECUTE the necessary processes to create an interprocess communication
     mechanism amoung the network nodes.  It is concerned with the set of
     services provided and management within the network.


PROCESS NAME:         EXECUTE NETWORK LEVEL PROTOCOL
PROCESS NUMBER:       1.5
PROCESS DESCRIPTION:
EXECUTE the processes necessary to control the routing of packets within the
     communication network.
```

PROCESS NAME: EXECUTE LINK LEVEL PROTOCOL
PROCESS NUMBER: 1.6
PROCESS DESCRIPTION:
EXECUTE the processes necessary to provide a data link control procedure that will ensure a reliable data communication path between nodes of an network.

PROCESS NAME: EXECUTE PHYSICAL LEVEL PROTOCOL
PROCESS NUMBER: 1.7
PROCESS DESCRIPTION:
EXECUTE the interface processes necessary to deliver a bit stream from one node to another.

PROCESS NAME: DECODE PROTOCOL RESPONSES
PROCESS NUMBER: 1.8
PROCESS DESCRIPTION:
DELIVER individual TRANSPORT_DISPLAY_COMMANDS to the operator as OPERATOR_DISPLAY

PROCESS NAME: EXECUTE EWNET PROTOCOL AT SECONDARY NODE
PROCESS NUMBER: 2.0
PROCESS DESCRIPTION:
EXECUTE the architecture protocol, transport protocol, network protocol, link protocol, and physical protocol established for the EWNET.

DATA DICTIONARY

FOR THE ARCHITECTURE LEVEL (A) PROTOCOL

Contents

DATA_FLOW_NAME: ABORT/STATUS_ERRORS
ALIASES: NONE
COMPOSITION:

```
ABORT/STATUS_ERRORS = | RECEIVED_UNSUPPORT_STATUS2 |
                      | RECEIVED_FORMAT_STATUS2    |
                      | RECEIVED_SYNC_STATUS2      |
                      | RECEIVED_UNSUPPORT_MISC    |
                      | RECEIVED_FORMAT_MISC       |
                      | RECEIVED_SYNC_UNKNOWN      |
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: ACC/ECF/SCF_PACKET
ALIASES: NONE
COMPOSITION:

```
ACC/ECF/SCF_PACKET = | ACCESS(ECF)_PACKET |
                     | ACCESS(SCF)_PACKET |
```

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: ACCEPT_CONFIRM
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG FROM THE TRANSPORT PROTOCOL IMPLYING THAT THE CONNECT_CONFIRM PACKET HAS BEEN RECEIVED AND THE LOGICAL_LINK ESTABLISHED.
NOTES: EXECUTE STARTUP PACKETS LAYER,
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: ACC/ER_PACKET
ALIASES: NONE
COMPOSITION:

```
ACC/ER_PACKET = | ACCESS(ERASE)_PACKET |
                | ACCESS(RENAME)_PACKET|
```

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ACCESS_COMPLETE_PACKET
ALIASES: RECEIVED_ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(PURGE)

RECEIVED_ACCOMP(RESPONSE)
RECEIVED_ACCOMP(EOS)

COMPOSITION:
ACCESS_COMPLETE_PACKET = OPERATOR + TYPE=7 + CMPFUNC + FOP

NOTES:
EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: ACCESS(ECF)_PACKET

ALIASES:
ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(RENAME)_PACKET
ACCESS(SCF)_PACKET
RECEIVED_ACCESS_PACKET
VALID_RECEIVED_ACCESS_PACKET

COMPOSITION:
SEE ALIASES

NOTES:
EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ACCESS(ERASE)_PACKET

ALIASES:
ACCESS_PACKET
ACCESS(RENAME)_PACKET
ACCESS(ECF)_PACKET
ACCESS(SCF)_PACKET
RECEIVED_ACCESS_PACKET
VALID_RECEIVED_ACCESS_PACKET

COMPOSITION:
SEE ALIASES

NOTES:
EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ACCESS_PACKET

ALIASES:
RECEIVED_ACCESS_PACKET
VALID_RECEIVED_ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(RENAME)_PACKET
ACCESS(ECF)_PACKET
ACCESS(SCF)_PACKET

COMPOSITION:
ACCESS_PACKET = OPERATOR + TYPE=3 + ACCFUNC + ACCOPT +
FILESPEC + FAC + SHR

NOTES:
EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: ACCESS(RENAME)_PACKET

ALIASES: ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(SCF)_PACKET
ACCESS(ECF)_PACKET
RECEIVED_ACCESS_PACKE.
VALID_RECEIVED_ACCESS_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ACCESS(SCF)_PACKET

ALIASES: ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(RENAME)_PACKET
ACCESS(ECF)_PACKET
RECEIVED_ACCESS_PACKET
VALID_RECEIVED_ACCESS_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: ACCFUNC

ALIASES: NONE

VALUES AND MEANINGS:
THE REQUEST CODE SPECIFYING THE OPERATION TO BE PERFORMED SUCH AS: OPEN FILE, CREATE FILE, RENAME FILE, ERASE FILE, DIRECTORY LIST, SUBMIT AS COMMAND FILE, AND EXECUTE COMMAND FILE.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: ACCOMP(COMMAND)

ALIASES: ACCOMP(PURGE)

VALUES AND MEANINGS:
A FLAG USED TO INDICATE THE END OF A DATA STREAM, PLUS A POSSIBLE ABORT, PLUS THE REQUESTING OF THE END OF THE LOGICAL LINK. GENERATES THE APPROPRIATE ACCESS_COMPLETE_PACKET

NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: ACCOMP(EOS)

ALIASES: NONE

VALUES AND MEANINGS:
A FLAG USED TO INDICATE THE END OF A DATA STREAM REQUEST.
GENERATES AN ACCESS_COMPLETE_PACKET.

NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: ACCOMP(PURGE)
ALIASES: ACCOMP(COMMAND)
VALUES AND MEANINGS:
SEE ALIASES

NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: ACCOPT
ALIASES: NONE
VALUES AND MEANINGS:
ACCESS OPTIONS:

1. A RECORD MAY BE SKIPPED OR REPEATED AS SPECIFIED BY THE CONTINUE PACKET. I/O ERRORS ARE NOT FATAL.
2. IF SET, A STATUS PACKET WILL BE RETURNED FOLLOWING EACH RECORD SENT TO THE ACCESSED PROCESS IN THE RECORD ACCESS MODE.
3. IF SET, RETURN A STATUS PACKET WITH EACH RECORD RETRIEVED FROM AN ACCESSED SYSTEM. THE STATUS PACKET SHOULD PRECEDE THE DATA PACKET SO THAT IT IS POSSIBLE TO BLOCK THE TWO INTO ONE PACKET. WHEN A USER REQUIRES A RECORD FILE ADDRESS TO BE RETURNED, THIS OPTION IS USED.

NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: ACKNOWLEDGE_PACKET
ALIASES: VALID_RECEIVED_ACKNOWLEDGE_PACKET
RECEIVED_ACKNOWLEDGE_PACKET
COMPOSITION:
ACKNOWLEDGE_PACKET = OPERATOR + TYPE=6

NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ADD_GET_PACKET
ALIASES: RECEIVED_ADD_GET_PACKET
COMPOSITION:

ADD_GET_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU + RAC + KEY + ROP

NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: ADD_PUT_PACKET
ALIASES: RECEIVED_ADD_PUT_PACKET
COMPOSITION:

ADD_PUT_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU + RAC + KEY + ROP

NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: ALQ
ALIASES: NONE
VALUES AND MEANINGS:

THIS FIELD SPECIFIES THE ALLOCATION QUANTITY. FOR FILE CREATION, IT SPECIFIES THE INITIAL SIZE OF THE NEW FILE. THE ACTUAL SIZE OF THE NEW FILE IS RETURNED IN THIS FIELD.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: ATTMENU
ALIASES: NONE
VALUES AND MEANINGS:

SPECIFIES WHICH OF THE ATTRIBUTES FIELDS WILL BE PRESENT IN THE MAIN ATTRIBUTES PACKET.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: ATTRIB/ACC_ERROR
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO NOTIFY THE ACCESS CHECK PROCESS THAT ON A RECEIVE OF A ATTRIB/ACC_PACKET THE ATTRIBUTES PORTION OF THE PACKET WAS IN ERROR.

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ATTRIB/ACC_PACKET
ALIASES: RECEIVED_ATTRIB/ACC_PACKET
COMPOSITION:

ATTRIB/ACC_PACKET = ATTRIBUTES_PACKET + ACCESS_PACKET

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: ATTRIB/ACK_ERROR
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO NOTIFY THE ACKNOWLEDGE CHECK PROCESS THAT ON A RECEIVE OF A ATTRIB/ACK_PACKET THE ATTRIBUTES PORTION OF THE PACKET WAS IN ERROR.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: ATTRIB/ACK_PACKET
ALIASES: RECEIVED_ATTRIB/ACK_PACKET
COMPOSITION:
ATTRIB/ACK_PACKET = ATTRIBUTES_PACKET + ACKNOWLEDGE_PACKET
NOTES: EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: ATTRIBUTES_PACKET
ALIASES: RECEIVED_ATTRIBUTES_PACKET
VALID_RECEIVED_ATTRIBUTES_PACKET
COMPOSITION:
ATTRIBUTES_PACKET = OPERATOR + TYPE=2 + ATTMENU + (DATATYPE) + (ORG) + (RFM) + (RAT) + (BLS) + (MRS) + (ALQ) + (BKS) + (FSZ) + (MRN) + (RUNSYS) + (DEQ) + (FOP)
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: BKS
ALIASES: NONE
VALUES AND MEANINGS:
BUCKET SIZE. USED ONLY FOR ACCESS TO RELATIVE FILES.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: BLS
ALIASES: NONE
VALUES AND MEANINGS:
PHYSICAL BLOCK SIZE.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: BUFSIZE
ALIASES: NONE
VALUES AND MEANINGS:

THE MAXIMUM BUFFER SIZE OF THE SENDING SYSTEM ALLOCATED FOR PACKET EXCHANGE. THE TWO SPEAKING PROCESSES WILL USE THE LESSER OF THE TWO BUFFER SIZES AS THE MAXIMUM SIZE. IF A SYSTEM HAS AN UNLIMITED BUFFER SIZE, IF SENDS A 0 AND THE TWO SYSTEMS WILL USE THE NONZERO BUFFER SIZE AS THE MAXIMUM.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: CMPFUNC
ALIASES: NONE
VALUES AND MEANINGS:

ACCESS COMPLETION FUNCTIONS SUCH AS:

1. TERMINATE ACCESS
2. RESPONSE
3. PURGE
4. END_OF_STREAM (EOS)

NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: CONFIGURATION_PACKET
ALIASES: RECEIVED_CONFIGURATION_PACKET
VALID_RECEIVED_CONFIGURATION_PACKET
COMPOSITION:

CONFIGURATION_PACKET = OPERATOR + TYPE=1 + BUFSIZE + OSTYPE + FILESYS + VERSION + SYSCAP

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: CONFUNC
ALIASES: NONE
VALUES AND MEANINGS:

THIS FIELD IS USED TO SPECIFY THE RECOVERY ACTION TO BE TAKEN.

1. TRY AGAIN
2. SKIP
3. ABORT

NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: CONNECT_PACKET
ALIASES: RECEIVED_CONNECT_PACKET
COMPOSITION:

CONNECT_PACKET = OPERATOR + TYPE=4 + CTLFUNC + (CTLMENU) + (RAC) + (KEY) + (ROP)

NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: CONTINUE_ABORT_ERROR
ALIASES: CONTINUE_SKIP_ERROR
CONTINUE_ONLY_ERROR
COMPOSITION:

```
CONTINUE_ABORT_ERROR = | RECEIVED_UNSUPPORT_CONTINUE |
                       | RECEIVED_FORMAT_CONTINUE    |
                       | RECEIVED_SYNC_CONTINUE      |
```

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTINUE_ABORT_PACKET
ALIASES: CONTINUE_ONLY_PACKET
CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_ONLY_PACKET
RECEIVED_CONTINUE_SKIP_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTINUE/DATA_PACKET
ALIASES: NONE
COMPOSITION:

```
CONTINUE/DATA_PACKET = CONTINUE_INTERRUPT_PACKET + DATA_
                       PACKET
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: CONTINUE_ERRORS
ALIASES: NONE
COMPOSITION:

```
CONTINUE_ERRORS = | RECEIVED_UNSUPPORT_CONTINUE |
                  | RECEIVED_FORMAT_CONTINUE    |
                  | RECEIVED_SYNC_CONTINUE      |
                  | RECEIVED_UNSUPPORT_MISC     |
                  | RECEIVED_FORMAT_MISC        |
                  | RECEIVED_SYNC_UNKNOWN       |
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: CONTINUE_INTERRUPT_PACKET
ALIASES: RECEIVED_CONTINUE_PACKETS
COMPOSITION:

CONTINUE_INTERRUPT_PACKET = CONTINUE_ABORT_PACKET +
CONTINUE_SKIP_PACKET +
CONTINUE_ONLY_PACKET

NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: CONTINUE_ONLY_ERROR
ALIASES: CONTINUE_ABORT_ERROR
CONTINUE_SKIP_ERROR

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTINUE_ONLY_PACKET
ALIASES: CONTINUE_ABORT_PACKET
CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ONLY_PACKET

COMPOSITION:
CONTINUE_ONLY_PACKET = OPERATOR + TYPE=5 + CONFUNC

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTINUE_SKIP_ERROR
ALIASES: CONTINUE_ABORT_ERROR
CONTINUE_ONLY_ERROR

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTINUE_SKIP_PACKET
ALIASES: CONTINUE_ONLY_PACKET
CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_ONLY_PACKET
RECEIVED_CONTINUE_SKIP_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: CONTINUE_WITH_BAD_RECORD

ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE BAD RECORD SHOULD BE LEFT AS IS AND CONTINUE RECIEVING RECORDS.
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: CONTROL_PACKET
ALIASES: RECEIVED_CONTROL_PACKET
COMPOSITION:

```
                  __                  __
CONTROL_PACKET = | CONNECT_PACKET     |
                 | GET_PACKET         |
                 | PUT_PACKET         |
                 --                  --
```

NOTES: GENERATE CONTROL PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: CTLFUNC
ALIASES: NONE
VALUES AND MEANINGS:
SPECIFIC CONTROL INFORMATION:

1. GET RECORD
2. CONNECT
3. UPDATE
4. PUT RECORD
5. DELETE
6. REWIND

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: CTLMENU
ALIASES: NONE
VALUES AND MEANINGS:
INDICATES OPTIONAL FIELDS ARE PRESENT.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: DATA
ALIASES: FILE_DATA
VALUES AND MEANINGS:
THE FILE DATA BEING TRANSFERRED. THIS FIELD IS TOTALLY TRANSPARENT AND USES ALL 8-BITS OF EACH BYTE.
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: DATA_ERRORS
ALIASES: NONE
COMPOSITION:

DATA_ERRORS = [RECEIVED_UNSUPPORT_DATA | RECEIVED_FORMAT_DATA | RECEIVED_SYNC_DATA | RECEIVED_UNSUPPORT_MISC | RECEIVED_FORMAT_MISC | RECEIVED_SYNC_UNKNOWN]

NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: DATA_PACKET
ALIASES: RECEIVED_DATA_FILE
COMPOSITION:

DATA_PACKET = OPERATOR + TYPE=8 + RECNUM + FILEDATA

NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: DATA_TYPE
ALIASES: NONE
VALUES AND MEANINGS:

THE TYPE OF DATA BEING TRANSFERRED. DEFAULT TO IMAGE. THIS FIELD IS VERY IMPORTANT FOR FILE/RECORD RETRIEVAL. MANY FILE SYSTEMS DO NOT EXPLICITLY STORE WITH THE FILE ATTRIBUTES, INFORMATION AS TO WHETHER THE FILE CONTAINS ASCII, EBCDIC, OR IMAGE DATA. THEREFORE, THE CONTENTS OF A FILE ARE INTERPRETED ACCORDING TO THE DATA TYPE SUPPLIED BY THE USER.
IMAGE IS THE MODE WHERE NO CODE SET IS SPECIFIED. IT IS A FORMAT FOR SENDING 8-BIT QUANTITIES WITHOUT SPECIFYING ANY CODE REPRESENTATION. THE ACTUAL DATA MAY BE ASCII, OR BINARY. IT IS UP TO THE USER PROCESS TO DETERMINE HOW TO USE THE DATA.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: DEQ
ALIASES: NONE
VALUES AND MEANINGS:

FILE EXTENSION QUANTUM SIZE IN VIRTUAL BLOCKS, WHICH IS THE AMOUNT OF SPACE, IN BLOCKS, ADDED TO THE FILE EACH TIME THE FILE IS IMPLICITLY EXTENDED.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: DISCONNECT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE TERMINATION OF THE LOGICAL LINK.
NOTES: CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: DISCONNECT_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: EOF
ALIASES: RECEIVED_EOF
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: EXECUTE_ERRORS
ALIASES: NONE
COMPOSITION:

```
                    __                          __
EXECUTE_ERRORS = | OPERATION_IN_PROGRESS |
                 | FILE_ERRORS           |
                 | SUCCESSFUL_OPERATION  |
                 | EOF                   |
                 | DATA_ERRORS           |
                 --                      --
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: FAC
ALIASES: NONE
VALUES AND MEANINGS:
FILE ACCESS OPERATIONS A USER REQUIRES.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FLAGS

ALIASES: NONE
VALUES AND MEANINGS:
THE PACKET FLAGS SUCH AS:
1. STREAM IDENTIFICATION FIELD PRESENT
2. LENGTH FIELD PRESENT
3. EXTENSION PRESENT
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FILE_DATA
ALIASES: DATA
VALUES AND MEANINGS:
SEE ALIASES
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FILE_ERRORS
ALIASES: RECEIVED_FILE_ERRORS
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: FILESPEC
ALIASES: NONE
VALUES AND MEANINGS:
THE FILE SPECIFICATION IN THE FORMAT REQUIRED BY THE REMOTE NODE.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FILESYS
ALIASES: NONE
VALUES AND MEANINGS:
FILE SYSTEM TYPE OF THE FILE SYSTEM BEING USED BY THE PROCESS SENDING THIS MESSAGE.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FIRST_ATTRIB_FLAG
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG SET WHEN THIS NODE IS THE ONE GENERATING THE FIRST ATTRIBUTES PACKET.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: FIRST_CONFIG_FLAG
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG SET WHEN THIS NODE IS THE ONE GENERATING THE FIRST CONFIGURATION_PACKET.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: FOP
ALIASES: NONE
VALUES AND MEANINGS:

FILE ACCESS OPTIONS A USER REQUIRES SUCH AS REWIND, OPEN, REWIND ON CLOSE, OR POSITION MAGNETIC TAPE, ETC.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: FSZ
ALIASES: NONE
VALUES AND MEANINGS:
SIZE OF THE FIXED PART OF VARIABLE LENGTH RECORDS WITH FIXED CONTROL FORMAT.
NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: GET_PACKET
ALIASES: RECEIVED_GET_PACKET
COMPOSITION:

```
GET_PACKET = | ADD_GET_PACKET |
             | KEY_GET_PACKET |
             | SEQ_GET_PACKET |
```

NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: INCOMING_PRIMARY_NODE_ARCHITECTURE_PACKETS
ALIASES: INCOMING_SECONDARY_NODE_ARCHITECTURE_PACKETS
COMPOSITION:
INCOMING_PRIMARY_NODE_ARCHITECTURE_PACKETS =

```
| RECEIVED_ACCESS_COMPLETE_PACKET |
| RECEIVED_STATUS_PACKET          |
| RECEIVED_STARTUP_PACKET         |
| RECEIVED_CONTINUE_ABORT         |
```

```
                                        | ACCEPT_CONFIRM                  |
                                        | RECEIVED_CONTINUE_ONLY_PACKET   |
                                        | RECEIVED_CONTINUE_SKIP_PACKET   |
                                        | RECEIVED_DATA_FILE              |
                                        | RECEIVED_CONTROL_FILE           |
                                        | DISCONNECT_CONFIRM_VERSION      |
                                        --                                --
NOTES:              EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
                    OVERVIEW LAYER


DATA FLOW NAME:     INCOMING_SECONDARY_NODE_ARCHITECTURE_PACKETS
ALIASES:            INCOMING_PRIMARY_NODE_ARCHITECTURE_PACKETS
COMPOSITION:
                    SEE ALIASES
NOTES:              OVERVIEW LAYER


DATA ELEMENT NAME:  KEY
ALIASES:            NONE
VALUES AND MEANINGS:
                    RELATIVE FILES = RECORD NUMBER
                    INDEXED FILES  = RECORD KEY
                    DIRECT FILES   = RECORD KEY
                    RECORD FILE ADDRESS:
                    ACCESS MODE    = RECORD FILE ADDRESS
                    BLOCK MODE ACC = VIRTUAL BLOCK NUMBER
NOTES:              ALL ARCHITECTURE LAYERS


DATA FLOW NAME:     KEY_GET_PACKET
ALIASES:            RECEIVED_KEY_GET_PACKET
COMPOSITION:
                    KEY_GET_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU +
                                     RAC + KEY + ROP
NOTES:              GENERATE CONTROL PACKETS LAYER


DATA FLOW NAME:     KEY_PUT_PACKET
ALIASES:            RECEIVED_KEY_PUT_PACKET
COMPOSITION:
                    KEY_PUT_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU +
                                     RAC + KEY + ROP
NOTES:              GENERATE CONTROL PACKETS LAYER
```

DATA ELEMENT NAME: LENGTH
ALIASES: NONE
VALUES AND MEANINGS:
DENOTES THE LENGTH OF THE OPERAND FIELD (NUMBER OF 8-BIT BYTES).
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: MRN
ALIASES: NONE
VALUES AND MEANINGS:
MAXIMUM RECORD NUMBER FOR A FILE.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: MRS
ALIASES: NONE
VALUES AND MEANINGS:
THE LENGTH OF EACH FILE RECORD IN NUMBER OF BYTES. FOR VARIABLE_LENGTH RECORDS, THIS FIELD SPECIFIES THE MAXIMUM RECORD SIZE. WHEN THE ACCESSED PROCESS RECEIVES THE MRS (MAXIMUM RECORD SIZE), IT MUST CHECK IT AGAINST; THE LENGTH OF ITS BUFFER. IF THE BUFFER WILL NOT ACCOMMODATE THIS SIZE RECORD, THE ACCESSED PROCESS SHOULD RETURN ITS BUFFER SIZE.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: NEW_ACCESS
ALIASES: NONE
VALUES AND MEANINGS:
A INTERNAL FLAG USED TO START A NEW DATA STREAM ACCESS ONCE AN OLD FILE TRANSFER IS COMPLETE.
NOTES: EXECUTE STARTUP PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: OPERAND
ALIASES: NONE
VALUES AND MEANINGS:
THE INFORMATION FIELD OF THE PACKET. IT IS DEPENDENT ON THE TYPE FIELD.
NOTES: ALL ARCHITECTURE LAYERS

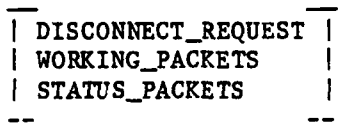
DATA ELEMENT NAME: OPERATION_IN_PROGRESS
ALIASES: RECEIVED_PENDING_STATUS
RECEIVED_OPERATION_IN_PROGRESS
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: OPERATOR
ALIASES: NONE
COMPOSITION:
OPERATOR = TYPE + FLAGS + (STREAMID) + (LENGTH) + (OPERAND)
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: ORG
ALIASES: NONE
VALUES AND MEANINGS:
ATTRIBUTES OF THE FILE BEING ACCESSED. DEFAULT = SEQUENTIAL.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: OSTYPE
ALIASES: NONE
VALUES AND MEANINGS:
OPERATING SYSTEM TYPE
NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: OUTGOING_PRIMARY_NODE_ARCHITECTURE_PACKETS
ALIASES: OUTGOING_SECONDARY_NODE_ARCHITECTURE_PACKETS
COMPOSITION:
OUTGOING_PRIMARY_NODE_ARCHITECTURE_PACKETS =



NOTES: OVERVIEW LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: OUTGOING_SECONDARY_NODE_ARCHITECTURE_PACKETS
ALIASES: OUTGOING_PRIMARY_NODE_ARCHITECTURE_PACKETS

COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: PRIMARY_INCOMING_DIALOGUE_COMMANDS
ALIASES: SECONDARY_INCOMING_DIALOGUE_COMMANDS
COMPOSITION:
PRIMARY_INCOMING_DIALOGUE_COMMANDS =

```
| OPERATOR_PASSWORD_COMMAND |
| OPERATOR_START_COMMAND    |
| ABORT_COMMAND             |
| CONNECT_REQUEST           |
```

NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: PRIMARY_START_FILE_CONFIGURATION
ALIASES: SECONDARY_START_FILE_CONFIGURATION
VALUES AND MEANINGS:
THIS IS A COMMAND FROM THE OPERATOR TO START THE FILE_ TRANSFER PROTOCOL BY ISSUING A CONFIGURATION_PACKET.
NOTES: OVERVIEW LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: PRIMARY_OUTGOING_DIALOGUE_COMMANDS
ALIASES: SECONDARY_OUTGOING_DIALOGUE_COMMANDS
COMPOSITION:
PRIMARY_OUTGOING_DIALOGUE_COMMANDS =

```
| RECEIVED_INCORRECT_PASSWORD_COMMAND |
| FLOW_CONTROL_ERRORS                 |
| OUTGOING_DIALOGUE_DATA_PACKET       |
| ERROR_REASON                        |
= OUTGOING_DIALOGUE_MESSAGE
```

REFER TO DATA DICTIONARY FOR TRANSPORT/NETWORK LEVEL PROTOCOL.
NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: PURGE/ABORT_INTERRUPT
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THAT A CONTINUE_ABORT_PACKET SHOULD BE GENERATED AND OUTPUTED AFTER THE GENERATION OF AN ACCOMP(PURGE)_PACKET OR ACCOMP(COMMAND)_PACKET.

NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: PUT_PACKET
ALIASES: RECEIVED_PUT_PACKET
COMPOSITION:

```
                    __                  __
PUT_PACKET = |  SEQ_PUT_PACKET  |
             |  KEY_PUT_PACKET  |
             |  ADD_PUT_PACKET  |
             --                  --
```

NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: RAC
ALIASES: NONE
VALUES AND MEANINGS:

SETS THE ACCESS MODE:

1. SEQUENTIAL RECORD ADDESS
2. KEYED ACCESS
3. ACCESS BY RECORD FILE ADDRESS
4. SEQUENTIAL FILE ACCESS
5. ACCESS BY VIRTUAL BLOCK NUMBER
6. *BLOCK MODE FILE TRANSFER*

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: RAT
ALIASES: NONE
VALUES AND MEANINGS:

INFORMATION ABOUT THE ATTRIBUTES OF THE INDIVIDUAL RECORDS SUCH AS TYPE OF CARRAGE CONTROL, LINE FEED, ETC.

NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: RECEIVED_ACCESS_COMPLETE_ERRORS
ALIASES: NONE
COMPOSITION:

RECEIVED_ACCESS_COMPLETE_ERRORS =

```
                                    __
                                   |  RECEIVED_UNSUPPORT_ACC/COMP  |
                                   |  RECEIVED_FORMAT_ACC/COMP     |
                                   |  RECEIVED_SYNC_ACC/COMP       |
                                   |  RECEIVED_UNSUPPORT_MISC      |
                                   |  RECEIVED_FORMAT_MISC         |
```

| RECEIVED_SYNC_UNKNOWN |

NOTES: EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACCESS_COMPLETE_PACKET
ALIASES: ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(PURGE)
RECEIVED_ACCOMP(RESPONSE)
RECEIVED_ACCOMP(EOS)
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_ACCESS_PACKET
ALIASES: ACCESS_PACKET
VALID_RECEIVED_ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(RENAME)_PACKET
ACCESS(ECF)_PACKET
ACCESS(SCF)_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACCOMP(COMMAND)
ALIASES: ACCESS_COMPLETE_PACKET
RECEIVED_ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(PURGE)
RECEIVED_ACCOMP(RESPONSE)
RECEIVED_ACCOMP(EOS)
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA_FLOW_NAME: RECEIVED_ACCOMP(EOS)
ALIASES: ACCESS_COMPLETE_PACKET
RECEIVED_ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET

RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(PURGE)
RECEIVED_ACCOMP(RESPONSE)

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACCOMP(PURGE)

ALIASES: ACCESS_COMPLETE_PACKET
RECEIVED_ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(RESPONSE)
RECEIVED_ACCOMP(EOS)

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACCOMP(RESPONSE)

ALIASES: ACCESS_COMPLETE_PACKET
RECEIVED_ACCESS_COMPLETE_PACKET
VALID_RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(PURGE)
RECEIVED_ACCOMP(EOS)

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACC_ERRORS

ALIASES: NONE

COMPOSITION:

RECEIVED_ACC_ERRORS = | RECEIVED_UNSUPPORT_ACCESS |
| RECEIVED_UNSUPPORT_MISC |
| RECEIVED_FORMAT_ACCESS |
| RECEIVED_FORMAT_MISC |
| RECEIVED_SYNC_ACCESS |
| RECEIVED_SYNC_UNKNOWN |

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACK_ERRORS

ALIASES: NONE

COMPOSITION:

RECEIVED_ACK_ERRORS = | RECEIVED_UNSUPPORT_MISC |
| RECEIVED_UNSUPPOET_ACK |
| RECEIVED_FORMAT_ACK |
| RECEIVED_FORMAT_MISC |
| RECEIVED_SYNC_ACK |
| RECEIVED_SYNC_UNKNOWN |

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ACKNOWLEDGE_PACKET
ALIASES: ACKNOWLEDGE_PACKET
VALID_RECEIVED_ACKNOWLEDGE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ADD_GET_PACKET
ALIASES: ADD_GET_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_ADD_PUT_PACKET
ALIASES: ADD_PUT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_ATTRIB/ACC_PACKET
ALIASES: ATTRIB/ACC_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ATTRIB/ACK_PACKET
ALIASES: ATTRIB/ACK_PACKET
COMPOSITION:
SEE ALIASES

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ATTRIB_ERRORS
ALIASES: NONE
COMPOSITION:

```
                                  ___                          ___
RECEIVED_ATTRIB_ERRORS = | RECEIVED_UNSUPPORT_ATTRIB |
                         | RECEIVED_FORMAT_ATTRIB    |
                         | RECEIVED_SYNC_ATTRIB      |
                         | RECEIVED_UNSUPPORT_MISC   |
                         | RECEIVED_FORMAT_MISC      |
                         | RECEIVED_SYNC_UNKNOWN     |
                         --                          --
```

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_ATTRIBUTES_PACKET
ALIASES: ATTRIBUTES_PACKET
VALID_RECEIVED_ATTRIBUTES_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_DATA_FILE
ALIASES: DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_CONFIG_ERRORS
ALIASES: NONE
COMPOSITION:

```
                                  ___                          ___
RECEIVED_CONFIG_ERRORS = | RECEIVED_UNSUPPORT_CONFIG |
                         | RECEIVED_FORMAT_CONFIG    |
                         | RECEIVED_SYNC_CONFIG      |
                         | RECEIVED_UNSUPPORT_MISC   |
                         | RECEIVED_FORMAT_MISC      |
                         | RECEIVED_SYNC_UNKNOWN     |
                         --                          --
```

NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONFIGURATION_PACKET

ALIASES: CONFIGURATION_PACKET
VALID_RECEIVED_CONFIGURATION_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONNECT_PACKET
ALIASES: CONNECT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTINUE_ABORT_PACKET
ALIASES: CONTINUE_ONLY_PACKET
CONTINUE_ABORT_PACKET
CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ONLY_PACKET
RECEIVED_CONTINUE_SKIP_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTINUE_ONLY_PACKET
ALIASES: CONTINUE_ONLY_PACKET
CONTINUE_SKIP_PACKET
CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ABORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTINUE_PACKETS
ALIASES: CONTINUE_INTERRUPT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_CONTINUE_SKIP_PACKET

ALIASES: CONTINUE_ONLY_PACKET
CONTINUE_ABORT_PACKET
CONTINUE_SKIP_PACKET
RECEIVED_CONTINUE_ABORT_PACKET
RECEIVED_CONTINUE_ONLY_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTINUE_STATUS
ALIASES: NONE
COMPOSITION:

RECEIVED_CONTINUE_STATUS =

| RECEIVED_OPERATION_IN_PROGRESS |
| RECEIVED_SUCCESSFUL_OPERATION |
| RECEIVED_SYNC_DATA |
| RECEIVED_FORMAT_DATA |
| RECEIVED_UNSUPPORT_DATA |
| RECEIVED_FILE_ERRORS |

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_CONTROL_ERRORS
ALIASES: NONE
COMPOSITION:

RECEIVED_CONTROL_ERRORS =

| RECEIVED_UNSUPPORT_CONTROL |
| RECEIVED_SYNC_CONTROL |
| RECEIVED_FORMAT_CONTROL |
| RECEIVED_UNSUPPORT_MISC |
| RECEIVED_FORMAT_MISC |
| RECEIVED_SYNC_UNKNOWN |

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE CONTROL PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTROL_PACKET
ALIASES: CONTROL_PACKET
COMPOSITION:
SEE ALIASES

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE CONTROL PACKETS LAYER

DATA FLOW NAME: RECEIVED_CONTROL_STATUS
ALIASES: NONE
COMPOSITION:

```
                    RECEIVED_CONTROL_STATUS = | RECEIVED_UNSUPPORT_CONTROL |
                                             | RECEIVED_FORMAT_CONTROL    |
                                             | RECEIVED_SYNC_CONTROL      |
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: RECEIVED_EOF
ALIASES: EOF
VALUES AND MEANINGS:

RECEIVED END-OF-FILE SUCCESSFULLY.

NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FILE_ERRORS
ALIASES: FILE_ERRORS
VALUES AND MEANINGS:

1. ERRORS THAT OCCUR BEFORE A FILE IS SUCCESSFULLY OPENED.
2. ERRORS THAT OCCUR AFTER OPENING A FILE AND BEFORE CLOSING THAT FILE.
3. ERRORS ASSOCIATED WITH TERMINATING ACCESS TO A FILE.

NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_FILE_STATUS
ALIASES: NONE
COMPOSITION:

```
                    RECEIVED_FILE_STATUS = | RECEIVED_FILE_ERRORS |
                                           | RECEIVED_TIMEOUT     |
```

NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_ACC/COMP
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:

SEE ALIASES

NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_ACCESS
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER

EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_ACK
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_ATTRIB
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES

NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_CONFIG
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_CONTINUE
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_CONTROL

ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_DATA
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_MISC
ALIASES: RECEIVED-FORMAT_CONFIG
RECEIVED-FORMAT_ATTRIB
RECEIVED-FORMAT_ACCESS
RECEIVED_FORMAT_CONTINUE
RECEIVED_FORMAT_CONTROL
RECEIVED_FORMAT_ACK
RECEIVED_FORMAT_ACC/ACK
RECEIVED_FORMAT_DATA
RECEIVED_FORMAT_STATUS2
VALUES AND MEANINGS:
1. ERROR IN PARSING THE PARTICULAR PACKET.
2. FIELD OF THE PARTICULAR PACKET IS INVALID (e.g., BITS THAT ARE MEANT TO BE MUTUALLY EXCLUSIVE ARE SET, AN UNDEFINED BIT IS SET, A FIELD VALUE IS OUT OF RANGE OR AN ILLEGAL STRING IS IN A FIELD).
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: RECEIVED_FORMAT_STATUS
ALIASES: NONE
COMPOSITION:

RECEIVED_FORMAT_STATUS = RECEIVED_FORMAT_MISC
RECEIVED_FORMAT_ACC/COMP
RECEIVED_FORMAT_CONFIG
RECEIVED_FORMAT_ATTRIB
RECEIVED_FORMAT_ACCESS
RECEIVED_FORMAT_CONTINUE
RECEIVED_FORMAT_CONTROL
RECEIVED_FORMAT_ACK
RECEIVED_FORMAT_DATA
RECEIVED_FORMAT_STATUS2

NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_FORMAT_STATUS2
ALIASES: RECEIVED_FORMAT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: RECEIVED_GET_PACKET
ALIASES: GET_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_KEY_GET_PACKET
ALIASES: KEY_GET_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_KEY_PUT_PACKET
ALIASES: KEY_PUT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_OPERATION_IN_PROGRESS
ALIASES: RECEIVED_PENDING_STATUS
OPERATION_IN_PROGRESS
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_PENDING_STATUS
ALIASES: RECEIVED_OPERATION_IN_PROGRESS
OPERATION_IN_PROGRESS
VALUES AND MEANINGS:
STATUS MESSAGE INDICATING THAT PAST SENT DATA PACKETS ARE BEING STORED/APPENDED, ETC. AND THAT ALL IS WELL.
NOTES: DECODE STATUS PACKETS LAYER

DATA FLOW NAME: RECEIVED_PUT_PACKET
ALIASES: PUT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_SEQ_GET_PACKET
ALIASES: SEQ_GET_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_SEQ_PUT_PACKET
ALIASES: SEP_PUT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_STARTUP_PACKETS
ALIASES: NONE
COMPOSITION:
RECEIVED_STARTUP_PACKETS =

| RECEIVED_CONFIGURATION_PACKET |
| RECEIVED_ACKNOWLEDGE_PACKET |
| RECEIVED_ATTRIB/ACK_PACKET |
| RECEIVED_ATTRIBUTES_PACKET |
| RECEIVED_ATTRIB/ACC_PACKET |
| RECEIVED_ACCESS_PACKET |

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: RECEIVED_STARTUP_STATUS
ALIASES: NONE
COMPOSITION:

```
                    RECEIVED_STARTUP_STATUS = | RECEIVED_UNSUPPORT_ACC/COMP |
                                             | RECEIVED_SYNC_CONFIG          |
                                             | RECEIVED_FORMAT_CONFIG        |
                                             | RECEIVED_UNSUPPORT_CONFIG     |
                                             | RECEIVED_FORMAT_ATTRIB        |
                                             | RECEIVED_FORMAT_ACCESS        |
                                             | RECEIVED_UNSUPPORT_ATTRIB     |
                                             | RECEIVED_SYNC_ATTRIB          |
                                             | RECEIVED_UNSUPPORT_ACCESS     |
                                             | RECEIVED_SYNC_ACCESS          |
                                             | RECEIVED_FORMAT_ACC/COMP      |
                                             | RECEIVED_SYNC_ACC/COMP        |
                                             | RECEIVED_UNSUPPORT_ACK        |
                                             | RECEIVED_SYNC_ACK             |
                                             | RECEIVED_FORMAT_ACK           |
                                             | RECEIVED_EOF                  |
                                             | RECEIVED_TIMEOUT              |
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_STATUS_ERROR
ALIASES: NONE
COMPOSITION:

```
                    RECEIVED_STATUS_ERROR = | RECEIVED_UNSUPPORT_STATUS |
                                            | RECEIVED_FORMAT_STATUS    |
                                            | RECEIVED_SYNC_STATUS      |
```

NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_STATUS_PACKET
ALIASES: VALID_RECEIVED_STATUS_PACKET
COMPOSITION:

```
                    RECEIVED_STATUS_PACKET = | RECEIVED_SUCCESSFUL_STATUS |
                                             | RECEIVED_UNSUPPORT_STATUS  |
                                             | RECEIVED_PENDING_STATUS    |
                                             | RECEIVED_FORMAT_STATUS     |
                                             | RECEIVED_SYNC_STATUS       |
                                             | RECEIVED_FILE_STATUS       |
```

NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SUCCESSFUL_OPERATION
ALIASES: SUCCESSFUL_OPERATION
VALUES AND MEANINGS:
RETURNS INFORMATION THAT INDICATES SUCCESS. USED WHEN PERFORMING RECORD-STORE ACTIONS.
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_SUCCESSFUL_STATUS
ALIASES: NONE
COMPOSITION:
RECEIVED_SUCCESSFUL_STATUS =

| RECEIVED_SUCCESSFUL_OPERATION |
| RECEIVED_EOF |

NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_ACC/COMP
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_ACCESS
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_ACK
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_ATTRIB
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_CONFIG
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_CONTINUE
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_CONTROL
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_DATA
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA FLOW NAME: RECEIVED_SYNC_STATUS
ALIASES: NONE
COMPOSITION:
RECEIVED_SYNC_STATUS = $\overline{|\ \text{RECEIVED\_SYNC\_CONFIG}\ \ |}$

RECEIVED_SYNC_ATTRIB
RECEIVED_SYNC_ACCESS
RECEIVED_SYNC_CONTROL
RECEIVED_SYNC_CONTINUE
RECEIVED_SYNC_ACK
RECEIVED_SYNC_ACC/COMP
RECEIVED_SYNC_DATA
RECEIVED_SYNC_STATUS2
RECEIVED_SYNC_UNKNOWN

NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_STATUS2
ALIASES: RECEIVED_SYNC_UNKNOWN
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_SYNC_UNKNOWN
ALIASES: RECEIVED_SYNC_CONFIG
RECEIVED_SYNC_ATTRIB
RECEIVED_SYNC_ACCESS
RECEIVED_SYNC_CONTROL
RECEIVED_SYNC_CONTINUE
RECEIVED_SYNC_ACK
RECEIVED_SYNC_ACC/COMP
RECEIVED_SYNC_DATA
RECEIVED_SYNC_STATUS2
VALUES AND MEANINGS:
THE PARTICULAR PACKET WAS RECEIVED OUT OF SYNCHRONIZATION.
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_TIMEOUT
ALIASES: TIMEOUT
VALUES AND MEANINGS:
A STATUS PACKET INDICATING THAT THE ACCESSED SYSTEM (NODE) RECEIVED A CONFIGURATION, ACCESS ATTRIBUTES, OR ACCESS_COMPLETE_PACKET AND HAS NOT RECEIVED THE APPROPRIATE PACKETS TO RESET THE TIMER. OR THE ACCESSED SYSTEM HAS RECEIVED A ACCESS_COMPLETE(RESPONSE) BUT AFTER A SUITABLE AMOUNT OF TIME NO OTHER COMMAND IS RECEIVED. WHEN A TIMEOUT IS SENT THE LOGICAL LINK IS ALSO TERMINATED.
NOTES: DECODE STATUS PACKETS LAYER

EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_ACC/COMP
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_ACCESS
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_ACK
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_ATTRIB
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_CONFIG
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_CONTINUE

ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_CONTROL
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_DATA
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_MISC
ALIASES: RECEIVED_UNSUPPORT_CONFIG
RECEIVED_UNSUPPORT_ATTRIB
RECEIVED_UNSUPPORT_ACCESS
RECEIVED_UNSUPPORT_CONTINUE
RECEIVED_UNSUPPORT_CONTROL
RECEIVED_UNSUPPORT_ACK
RECEIVED_UNSUPPORT_ACC/COMP
RECEIVED_UNSUPPORT_DATA
RECEIVED_UNSUPPORT_STATUS2
VALUES AND MEANINGS:
THIS IS USED WHEN AN UNSUPPORTED BIT/FIELD OR A FIELD/
VALUE, FOR A PARTICULAR PACKET TYPE, IS ENCOUNTERED WHICH
A PARTICULAR IMPLEMENTATION DOES NOT SUPPORT.
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: RECEIVED_UNSUPPORT_STATUS
ALIASES: NONE
COMPOSITION:

RECEIVED_UNSUPPORT_STATUS = [RECEIVED_UNSUPPORT_CONTINUE]
[RECEIVED_UNSUPPORT_MISC]
[RECEIVED_UNSUPPORT_CONFIG]

| RECEIVED_UNSUPPORT_ATTRIB |
| RECEIVED_UNSUPPORT_ACCESS |
| RECEIVED_UNSUPPORT_CONTROL |
| RECEIVED_UNSUPPORT_ACK |
| RECEIVED_UNSUPPORT_ACC/COMP |
| RECEIVED_UNSUPPORT_DATA |
| RECEIVED_UNSUPPORT_STATUS2 |

NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: RECEIVED_UNSUPPORT_STATUS2
ALIASES: RECEIVED_UNSUPPORT_MISC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: RECNUM
ALIASES: NONE
VALUES AND MEANINGS:
THIS FIELD IS USED TO SEND THE RECORD NUMBER WHEN ACCESSING RELATIVE FILES. FOR RANDOM STORE, THIS FIELD WILL CONTAIN THE RECORD NUMBER. WHEN IN THE BLOCK MODE, THIS FIELD WILL CONTAIN THE VIRTUAL BLOCK NUMBER.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: RESEND_INFO
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG TO INDICATE THAT THE BAD RECORD SHOULD BE RESENT AND THEN CONTINUE ON UNTIL EOF.
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: RFA
ALIASES: NONE
VALUES AND MEANINGS:
USED TO RETURN THE RECORD FILE ADDRESS OF THE RECORD TO WHICH THIS STATUS PACKET APPLIES.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: RFM

ALIASES: NONE
VALUES AND MEANINGS:
FORMAT OF THE RECORDS BEING TRANSFERRED.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: ROP
ALIASES: NONE
VALUES AND MEANINGS:
OPTIONAL RECORD PROCESSING FEATURES SUCH AS POSITION OF EOF.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: RUNSYS
ALIASES: NONE
VALUES AND MEANINGS:
NAME OF THE RUN-TIME SYSTEM ENVIROMENT REQUIRED TO EXECUTE THE CODE CONTAINED IN THE FILE. THIS FIELD IS USEFUL TO TO OPERATING SYSTEMS THAT CAN EMULATE OTHER OPERATING SYSTEM ENVIROMENTS.
NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: SECONDARY_INCOMING_DIALOGUE_COMMANDS
ALIASES: PRIMARY_INCOMING_DIALOGUE_COMMANDS
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_DIALOGUE_COMMANDS
ALIASES: PRIMARY_OUTGOING_DIALOGUE_COMMANDS
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: SECONDARY_START_FILE_CONFIGURATION
ALIASES: PRIMARY_START_FILE_CONFIGURATION
VALUES AND MEANINGS:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: SECOND_ATTRIB_FLAG
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THAT THIS NODE HAS NOT PREVIOUSLY ISSUED AN ATTRIBUTES PACKET BUT HAS RECEIVED ONE.

NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: SECOND_CONFIG_FLAG
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THAT THIS NODE HAS NOT PREVIOUSLY ISSUED A CONFIGURATION_PACKET BUT HAS RECEIVED ONE.

NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: SEQ_GET_PACKET
ALIASES: RECEIVED_SEQ_GET_PACKET
COMPOSITION:

SEQ_GET_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU + RAC + ROP

NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: SEQ_PUT_PACKET
ALIASES: RECEIVED_SEQ_PUT_PACKET
COMPOSITION:

SEQ_PUT_PACKET = OPERATOR + TYPE=4 + CTLFUNC + CTLMENU + RAC + ROP

NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: SETUP_ERRORS
ALIASES: NONE
COMPOSITION:

SETUP_ERRORS = | RECEIVED_ACCESS_COMPLETE_ERRORS |
| RECEIVED_CONFIG_ERRORS |
| RECEIVED_ACK_ERRORS |
| RECEIVED_ATTRIB_ERRORS |
| RECEIVED_ACC_ERRORS |

NOTES: EXECUTE STARTUP PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA FLOW NAME: SETUP_PACKETS
ALIASES: NONE
COMPOSITION:

```
SETUP_PACKETS = | ACCESS_COMPLETE_PACKET |
                | ATTRIB/ACK_PACKET       |
                | ACKNOWLEDGE_PACKET      |
                | ACCESS_PACKET           |
                | ATTRIB/ACC_PACKET       |
                | ACC/ER_PACKET           |
                | ACC/ECF/SCF_PACKET      |
                | ATTRIBUTES_PACKET       |
```

NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: SHR

ALIASES: NONE
VALUES AND MEANINGS:
OPERATIONS SHARED WITH OTHER USERS.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: SKIP_INFO/CONTINUE
ALIASES: SKIP_REC/CONTINUE
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: SKIP_REC/CONTINUE
ALIASES: SKIP_INFO/CONTINUE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE BAD RECORD SHOULD BE SKIPPED OVER AND THE REMAINING RECORDS PROCESSED.
NOTES: EXECUTE CONTINUE PACKETS LAYER

DATA ELEMENT NAME: START_ACC/COMP
ALIASES: START_ER/ECF_COMP
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: START_ACC_ECF
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START GENERATION OF AN ACCESS(ECF) OR ACCESS(SCF) PACKETS.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: START_ACC_ER
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THE GENERATION OF AN ACCESS(ERASE) PACKET, OR ACCESS(RENAME) PACKET.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: START_ACCESS
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE LAST ATTRIBUTES PACKET HAS BEEN RECEIVED AND A ACCESS PACKET CAN NOW BE GENERATED AND OUTPUTTED.
NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: START_ATTRIB/ACC
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THE GENERATION OF A ATTRIBUTES/ ACCESS PACKET (ATTRIB/ACC_PACKET).
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: START_ACK
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF AN ACKNOWLEDGE_ PACKET.
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: START_ADD_GET
ALIASES: NONE

VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL ADD_GET_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_ADD_PUT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL ADD_PUT_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_ATTRIB/ACK
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF AN ATTRIB/ACK_PACKET.
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: START_ATTRIBUTES
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG WHICH IS ISSUED WHEN FIRST_CONFIGURATION_FLAG IS *SET AND WE RECEIVE A CONFIGURATION PACKET.* IT STARTS GENERATION OF THE FIRST ATTRIBUTES_PACKET.
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA ELEMENT NAME: START_CONNECT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL_CONNECT_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_CONNECT_ACK
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG MEANING THAT A VALID CONTROL_CONNECT_PACKET HAS BEEN RECEIVED, THEREFORE ISSUE AN ACKNOWLEDGE_PACKET.
NOTES: EXECUTE ACC/ACK PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER
EXECUTE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_ER/ECF_COMP
ALIASES: START_ACC/COMP
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF AN ACCESS_COMPLETE_ PACKET.
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA ELEMENT NAME: START_KEY_GET
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL KEY_GET_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_KEY_PUT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL KEY_PUT_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_SEQ_PUT_APPEND
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL SEQ_PUT_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA ELEMENT NAME: START_SEQ_GET
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START THE GENERATION OF A CONTROL SEQ_GET_ PACKET.
NOTES: GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: STARTUP_ERRORS
ALIASES: NONE
COMPOSITION:

```
                    STARTUP_ERRORS = RECEIVED_ACCESS_COMPLETE_ERRORS
NOTES:              EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER


DATA FLOW NAME:     STARTUP_PACKETS
ALIASES:            NONE
COMPOSITION:
                    STARTUP_PACKETS = | ACCESS_COMPLETE_PACKET |
                                      | ATTRIB/ACK_PACKET      |
                                      | ACKNOWLEDGE_PACKET     |
                                      | ACCESS_PACKET          |
NOTES:              EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER


DATA FLOW NAME:     STATUS_PACKETS
ALIASES:            NONE
COMPOSITION:
                    STATUS_PACKETS = | RECEIVED_CONTROL_ERRORS     |
                                     | RECEIVED_UNSUPPORT_STATUS2 |
                                     | RECEIVED_FORMAT_STATUS2    |
                                     | RECEIVED_STATUS_ERROR      |
                                     | SETUP_ERRORS               |
                                     | EOF                        |
                                     | FILE_ERRORS                |
                                     | SUCCESSFUL_OPERATION       |
                                     | OPERATION_IN_PROGRESS      |
                                   = OPERATOR + TYPE=9 + STSCODE + RFA +
                                     RECNUM + STV
NOTES:              CODE ARCHITECTURE PACKETS LAYER


DATA ELEMENT NAME:  STREAMID
ALIASES:            NONE
VALUES AND MEANINGS:
                    THE STREAM IDENTIFICATION FIELD.  THIS FIELD IS USED TO
                    ALLOW A SINGLE USER TO HAVE MULTIPLE DATA STREAMS IN USE
                    FOR A SINGLE OPEN FILE.  ALL DATA STREAMS USE THE SAME
                    LOGICAL LINK (MULTIPLEX ON THE STEAMID NUMBER).
NOTES:              ALL ARCHITECTURE LAYERS


DATA ELEMENT NAME:  STSCODE
ALIASES:            NONE
VALUES AND MEANINGS:
                    STATUS FIELD = MACCODE = THE MACRO OR FUNCTIONAL GROUP
```

REASON FOR THE ERROR.
MICCODE = THE SPECIFIC REASON FOR THE ERROR.

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: STV
ALIASES: NONE
VALUES AND MEANINGS:
SECONDARY STATUS. USED TO RETURN SECONDARY STATUS INFORMATION SUCH AS DEVICE ERROR CODES, ETC.
NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: SUBMIT_AS_COMMAND_FILE
ALIASES: NONE
VALUES AND MEANINGS:
REQUEST IN THE ACCESS_PACKET THAT REQUESTS THAT A STORE OPERATION BE DONE ON THE DATA THAT FOLLOWS IN A TEMPORARY FILE AND THAT THIS FILE BE SUBMITTED TO A BATCH-TYPE FACILITY UPON ACCESS COMPLETION. THE FILE WILL BE DELETED FOLLOWING EXECUTION BY THE BATCH FACILITY. THE FILE IS TRANSFERRED USING SEQUENTIAL FILE STORAGE.
NOTES: EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: SUCCESSFUL_OPERATION
ALIASES: RECEIVED_SUCCESSFUL_OPERATION
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE CONTINUE PACKETS LAYER
CODE ARCHITECTURE PACKETS LAYER

DATA ELEMENT NAME: SYSCAP
ALIASES: NONE
VALUES AND MEANINGS:
GENERIC SYSTEM CAPABILITIES SUCH AS;
1. SUPPORTS SEQUENTIAL FILE ORGANIZATION
2. SUPPORTS RELATIVE FILE ORGANIZATION
3. SUPPORTS FILE PREALLOCATION
4. SUPPORTS SEQUENTIAL FILE ACCESS
5. SUPPORTS RANDOM ACCESS BY RECORD NUMBER
6. SUPPORTS RANDOM ACCESS BY VIRTUAL BLOCK NUMBER
7. SUPPORTS APPEND TO FILE ACCESS
8. SUPPORTS COMMAND FILE SUBMISSION AND/OR EXECUTION
9. SUPPORTS MULTIPLE DATA STREAMS
10. SUPPORTS STATUS RETURN AND

11. SUPPORST BLOCKING OF PACKETS

NOTES: ALL ARCHITECTURE LAYERS

DATA ELEMENT NAME: TIMEOUT
ALIASES: RECEIVED_TIMEOUT
VALUES AND MEANINGS:
SEE ALIASES
NOTES: CODE ARCHITECTURE PACKETS

DATA ELEMENT NAME: TYPE
ALIASES: NONE
VALUES AND MEANINGS:
THE TYPE OF PACKET. REFER TO EACH INDIVIDUAL MESSAGE FOR ITS TYPE.
NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: VALID_RECEIVED_ACCESS_COMPLETE_PACKET
ALIASES: ACCESS_COMPLETE_PACKET
RECEIVED_ACCESS_COMPLETE_PACKET
RECEIVED_ACCOMP(COMMAND)
RECEIVED_ACCOMP(PURGE)
RECEIVED_ACCOMP(RESPONSE)
RECEIVED_ACCOMP(EOS)
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ACC/ACK PACKETS LAYER

DATA FLOW NAME: VALID_RECEIVED_ACCESS_PACKET
ALIASES: ACCESS_PACKET
RECEIVED_ACCESS_PACKET
ACCESS(ERASE)_PACKET
ACCESS(RENAME)_PACKET
ACCESS(ECF)_PACKET
ACCESS(SCF)_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER
EXECUTE ACC/ACK PACKETS LAYER
EXECUTE ARCHITECTURE PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: VALID_RECEIVED_ACKNOWLEDGE_PACKET

ALIASES: ACKNOWLEDGE_PACKET
RECEIVED_ACKNOWLEDGE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER
GENERATE CONTROL PACKETS LAYER

DATA FLOW NAME: VALID_RECEIVED_ATTRIBUTES_PACKET
ALIASES: ATTRIBUTES_PACKET
RECEIVED_ATTRIBUTES_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: VALID_RECEIVED_CONFIGURATION_PACKET
ALIASES: CONFIGURATION_PACKET
RECEIVED_CONFIGURATION_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKETS LAYER

DATA FLOW NAME: VALID_RECEIVED_STATUS_PACKET
ALIASES: RECEIVED_STATUS_PACKET
COMPOSITION:
SEE ALIASES
NOTES: DECODE STATUS PACKETS LAYER

DATA ELEMENT NAME: VERSION
ALIASES: NONE
VALUES AND MEANINGS:
A FIELD IDENTIFYING THE PROTOCOL AND SOFTWARE VERSION NUMBERS.
NOTES: ALL ARCHITECTURE LAYERS

DATA FLOW NAME: WORKING_PACKETS
ALIASES: NONE
COMPOSITION:

```
WORKING_PACKETS = | SETUP_PACKETS             |
                  | DATA_PACKETS              |
                  | CONTROL_PACKETS           |
                  | CONTINUE_INTERRUPT_PACKET |
```

NOTES:

CODE ARCHITECTURE PACKETS LAYER

FILE DEFINITIONS (A)

FILE OR DATABASE NAME:	LOCAL_DATA_FILE
ALIASES:	NONE
COMPOSITION:	
	THE MEMORY USED TO READ, WRITE, AND DELETE FILES AT THIS NODE.
ORGANIZATION:	STANDARD MEMORY
NOTES:	EXECUTE CONTINUE PACKETS LAYER

PROCESS SPECIFICATION (A)

```
PROCESS NAME:          DECODE ARCHITECTURE PACKETS
PROCESS NUMBER:        1.1
PROCESS DESCRIPTION:
IF Input type code = 9 then
     Output as RECEIVED_STATUS_PACKET
ELSEIF Input type code = 8 then
     Output as RECEIVED_DATA_FILE
ELSEIF Input type code = 7 then
     Output as RECEIVED_ACCESS_COMPLETE_PACKET
ELSEIF Input type code = 1, 2, 3, or 6 then
     Output as RECEIVED_STARTUP_PACKETS
ELSEIF Input type code = 5 then
     Output as RECEIVED_CONTINUE_PACKETS
ELSEIF Input type code = 4 then
     Output as RECEIVED_CONTROL_PACKETS
ELSE output as ACCEPT_CONFIRM
```

```
PROCESS NAME:          CHECK FOR ERRORS
PROCESS NUMBER:        1.2.1
PROCESS DESCRIPTION:
ON a RECEIVED_STATUS_PACKET
     IF errors exist in packet then
          Output as RECEIVED_STATUS_ERROR
     ELSE output as VALID_RECIEVED_STATUS_PACKET
```

```
PROCESS NAME:          DECODE MACCODE FIELD
PROCESS NUMBER:        1.2.2
PROCESS DESCRIPTION:
IF Input MACCODE value = 0 then
     Output as RECEIVED_PENDING_STATUS
ELSEIF Input MACCODE value =1 then
     Output as RECEIVED_SUCCESSFUL_STATUS
ELSEIF Input MACCODE value = 2 then
     Output as RECEIVED_UNSUPPORTED_STATUS
ELSEIF Input MACCODE value = 4 then
     Output as RECEIVED_FILE_STATUS
ELSEIF Input MACCODE value = 5 then
     Output as RECEIVED_FORMAT_STATUS
ELSE Input MACCODE value = 6 then
     Output as RECEIVED_SYNC_STATUS
```

```
PROCESS NAME:          DECODE UNSUPPORTED MICCODE FIELD
PROCESS NUMBER:        1.2.3
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
     Output as RECEIVED_UNSUPPORT_MISC
ELSEIF Input MICCODE value = 1 then
     Output as RECEIVED_UNSUPPORT_CONFIG
ELSEIF Input MICCODE value = 2 then
     Output as RECIEVED_UNSUPPORT_ATTRIB
ELSEIF Input MICCODE value = 3 then
     Output as RECEIVED_UNSUPPORT_ACCESS
ELSEIF Input MICCODE value = 4 then
     Output as RECEIVED_UNSUPPORT_CONTINUE
ELSEIF Input MICCODE value = 5 then
     Output as RECEIVED_UNSUPPORT_CONTROL
ELSEIF Input MICCODE value = 6 then
     Output as RECEIVED_UNSUPPORT_ACK
ELSEIF Input MICCODE value = 7 then
     Output as RECEIVED_UNSUPPORT_ACC/COMP
ELSEIF Input MICCODE value = 8 then
     Output as RECEIVED_UNSUPPORT_DATA
ELSE Input MICCODE value = 9 then
     Output as RECEIVED_UNSUPPORT_STATUS2


PROCESS NAME:          DECODE PENDING MICCODE FIELD
PROCESS NUMBER:        1.2.4
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
     Output as RECEIVED_OPERATION_IN_PROGRESS
ELSE Null


PROCESS NAME:          DECODE FORMAT MICCODE FIELD
PROCESS NUMBER:        1.2.5
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
     Output as RECEIVED_FORMAT_MISC
ELSEIF Input MICCODE value = 1 then
     Output as RECEIVED_FORMAT_CONFIG
ELSEIF Input MICCODE value = 2 then
     Output as RECEIVED_FORMAT_ATTRIB
ELSEIF Input MICCODE value = 3 then
     Output as RECEIVED_FORMAT_ACCESS
ELSEIF Input MICCODE value = 4 then
     Output as RECEIVED_FORMAT_CONTINUE
ELSEIF Input MICCODE value = 5 then
     Output as RECEIVED_FORMAT_CONTROL
```

```
ELSEIF Input MICCODE value = 6 then
     Output as RECEIVED_FORMAT_ACK
ELSEIF Input MICCODE value = 7 then
     Output as RECEIVED_FORMAT_ACC/COMP
ELSEIF Input MICCODE value = 8 then
     Output as RECEIVED_FORMAT_DATA
ELSE Input MICCODE value = 9 then
     Output as RECEIVED_FORMAT_STATUS2


PROCESS NAME:          DECODE FILE MICCODE FIELD
PROCESS NUMBER:        1.2.6
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
     Output as RECEIVED_FILE_ERRORS
ELSE Input MICCODE value = 1 then
     Output as RECEIVED_TIMEOUT


PROCESS NAME:          DECODE SYNC MICCODE FIELD
PROCESS NUMBER:        1.2.7
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
     Output as RECEIVED_SYNC_CONFIG
ELSEIF Input MICCODE value = 1 then
     Output as RECEIVED_SYNC_ATTRIB
ELSEIF Input MICCODE value = 2 then
     Output as RECEIVED_SYNC_ACCESS
ELSEIF Input MICCODE value = 3 then
     Output as RECEIVED_SYNC_CONTROL
ELSEIF Input MICCODE value = 4 then
     Output as RECEIVED_SYNC_CONTINUE
ELSEIF Input MICCODE value = 5 then
     Output as RECEIVED_SYNC_ACK
ELSEIF Input MICCODE value = 6 then
     Output as RECEIVED_SYNC_ACC/COMP
ELSEIF Input MICCODE value = 7 then
     Output as RECEIVED_SYNC_DATA
ELSEIF Input MICCODE value = 8 then
     Output as RECEIVED_SYNC_STATUS2
ELSE Input MICCODE value = 9 then
     Output as RECEIVED_SYNC_UNKNOWN


PROCESS NAME:          DECODE SUCCESSFUL MICCODE FIELD
PROCESS NUMBER:        1.2.8
PROCESS DESCRIPTION:
IF Input MICCODE value = 0 then
```

```
     Output as RECIEVED_EOF
ELSE Input MICCODE value = 1 then
     Output as RECEIVED_SUCCESSFUL_OPERATION


PROCESS NAME:          DECODE CONTROL TYPE
PROCESS NUMBER:        1.3.1
PROCESS DESCRIPTION:
IF Input = VALID_RECEIVED_ACKNOWLEDGE_PACKET and connect flag is not set then
     Output a START_CONNECT
          Set the connect flag
ELSEIF Input = VALID_RECEIVED_ACKNOWLEDGE_PACKET and connect flag is set then
     Generate the following outputs according to the file process to be
     performed:
          Reset the connect flag
                  FILE-PROCESS                          OUTPUT
                  ------------                          ------

              Sequential file storage              START_SEQ_PUT_APPEND
              Sequential file append               START_SEQ_PUT_APPEND
              Sequential file retrieval            START_SEQ_GET
              Keyed record retrieval               START_KEY_GET
              Record file address retrieval        START_ADD_GET
              Keyed record storage                 START_KEY_PUT
              Record file address storage          START_ADD_PUT
              Submit as command file               START_SEQ_PUT_APPEND
ELSE Input = STATUS_PACKET then
     Output above start command depending on the value passed in the Scatus_
     Packet MACCODE, MICCODE field combination


PROCESS NAME:          GENERATE CONTROL CONNECT PACKET
PROCESS NUMBER:        1.3.2
PROCESS DESCRIPTION:
IF Input = START_CONNECT then
     Output CONNECT_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTROL SEQ-GET PACKET
PROCESS NUMBER:        1.3.3
PROCESS DESCRIPTION:
IF Input = START_SEQ_GET then
     Output SEQ_GET_PACKET
ELSE Null
```

```
PROCESS NAME:          GENERATE CONTROL SEQ-PUT PACKET
PROCESS NUMBER:        1.3.4
PROCESS DESCRIPTION:
IF Input = START_SEQ_PUT_APPEND then
     Output SEQ_PUT_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTROL KEY-GET PACKET
PROCESS NUMBER:        1.3.5
PROCESS DESCRIPTION:
IF Input = START_KEY_GET then
     Output KEY_GET_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTROL KEY-PUT PACKET
PROCESS NUMBER:        1.3.6
PROCESS DESCRIPTION:
IF Input = START_KEY_PUT then
     Output KEY_PUT_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTROL ADD-GET PACKET
PROCESS NUMBER:        1.3.7
PROCESS DESCRIPTION:
IF Input = START_ADD_GET then
     Output ADD_GET_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTROL ADD-PUT PACKET
PROCESS NUMBER:        1.3.8
PROCESS DESCRIPTION:
IF Input = START_ADD_PUT then
     Output ADD_PUT_PACKET
ELSE Null


PROCESS NAME:          CODE GET FIELDS
PROCESS NUMBER:        1.3.9
PROCESS DESCRIPTION:
IF Input = SEQ_GET_PACKET, KEY_GET_PACKET, or ADD_GET PACKET then
     Output as GET_PACKET
ELSE Null
```

```
PROCESS NAME:          CODE PUT FIELDS
PROCESS NUMBER:        1.3.10
PROCESS DESCRIPTION:
IF Input = SEQ_PUT_PACKET, KEY_PUT_PACKET, or ADD_PUT_PACKET then
     Output as PUT_PACKET
ELSE Null


PROCESS NAME:          CODE CONTROL FIELDS
PROCESS NUMBER:        1.3.11
PROCESS DESCRIPTION:
IF Input = GET_PACKET or PUT_PACKET then
     Output as CONTROL_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONFIGURATION PACKET
PROCESS NUMBER:        1.4.1
PROCESS DESCRIPTION:
IF Input = PRIMARY_START_FILE_CONFIGURATION and ACCEPT_CONFIRM or input =
     PRIMARY_START_FILE_CONFIGURATION and NEW_ACCESS then
          Output CONFIGURATION_PACKET and
          Output FIRST_CONFIG_FLAG
ELSE Input = RECEIVED_SYNC_CONFIG, RECEIVED_TIMEOUT, RECEIVED_FORMAT_CONFIG,
     RECEIVED_UNSUPPORT_CONFIG, RECEIVED_FORMAT_ATTRIB, RECEIVED_FORMAT_
     ACCESS, or SECOND_CONFIG_FLAG then
          Output CONFIGURATION_PACKET


PROCESS NAME:          CHECK FOR CONFIGURATION ERRORS
PROCESS NUMBER:        1.4.2
PROCESS DESCRIPTION:
IF Input contains errors then
     Output RECEIVED_CONFIG_ERRORS
ELSE output VALID_RECEIVED_CONFIGURATION_PACKET


PROCESS NAME:          DECODE CONFIGURATION FIELDS
PROCESS NUMBER:        1.4.3
PROCESS DESCRIPTION:
IF Input SYSCAP value = ERASE or RENAME then
     Output START_ACC_ER flag
ELSEIF Input SYSCAP value = EXECUTE_COMMAND_FILE or SUBMIT_COMMAND_FILE then
     Output START_ACC_ECF flag
```

```
ELSEIF FIRST_CONFIG_FLAG is set and BUFSIZ is large enough then
     Output START_ATTRIB/ACC flag
     Reset FIRST_CONFIG_FLAG
ELSEIF FIRST_CONFIG_FLAG is set and BUFSIZ is small then
     Output START_ATTRIBUTES flag
     Reset FIRST_CONFIG_FLAG
ELSE FIRST_CONFIG_FLAG is not set then
     Output a SECOND_CONFIG_FLAG



PROCESS NAME:          GENERATE ATTRIBUTES/ACCESS PACKET
PROCESS NUMBER:        1.4.4
PROCESS DESCRIPTION:
IF Input = START_ATTRIB/ACC or RECEIVED_TIMEOUT then
     Output a ATTRIB/ACC_PACKET
ELSE Null



PROCESS NAME:          GENERATE ACCESS ERASE, RENAME PACKET
PROCESS NUMBER:        1.4.5
PROCESS DESCRIPTION:
IF Input = START_ACC_ER or RECEIVED_TIMEOUT and ERASE is indicated then
     Output an ACC/ER_PACKET to erase
ELSE Input = START_ACC_ER or RECEIVED_TIMEOUT and RENAME is indicated then
     Output an ACC/ER_PACKET to rename



PROCESS NAME:          GENERATE ACCESS ECF/SCF PACKET
PROCESS NUMBER:        1.4.6
PROCESS DESCRIPTION:
IF Input = START_ACC_ECF/SCF or RECEIVED_TIMEOUT and ECF is indicated then
     Output an EXECUTE_COMMAND_FILE ACC/ECF/SCF_PACKET
ELSE Input = START_ACC_ECF or RECEIVED_TIMEOUT and SCF is indicated then
     Output a SUBMIT COMMAND FILE  ACC/ECF/SCF_PACKET



PROCESS NAME:          GENERATE ATTRIBUTES PACKET
PROCESS NUMBER:        1.4.7
PROCESS DESCRIPTION:
IF Input = START_ATTRIBUTES then
     Output FIRST_ATTRIB_FLAG
     Output ATTRIBUTES_PACKET
ELSEIF Input = SECOND_ATTRIB_FLAG then
     Output ATTRIBUTES_PACKET
ELSE Input = RECEIVED_UNSUPPORT_ATTRIB, RECEIVED_SYNC_ATTRIB, or RECEIVED_
     TIMEOUT then
     Output ATTRIBUTES_PACKET
```

```
     Reset FIRST_ATTRIB_FLAG


PROCESS NAME:          CHECK FOR ACKNOWLEDGE ERRORS
PROCESS NUMBER:        1.4.8
PROCESS DESCRIPTION:
IF Input = RECEIVED_ACKNOWLEDGE_PACKET and no errors exist then
     Output VALID_RECEIVED_ACKNOWLEDGE_PACKET
ELSEIF Input = RECEIVED_ACKNOWLEDGE_PACKET with errors then
     Output RECEIVED_ACK_ERRORS
ELSEIF Input = RECEIVED_ATTRIB/ACK_PACKET and errors exist then
     Output RECEIVED_ACK_ERRORS
ELSEIF Input = RECEIVED_ATTRIB/ACK_PACKET and no errors exist and ATTRIB/ACK_
     ERROR_FLAG is not set then
          Output VALID_RECEIVED_ACKNOWLEDGE_PACKET
ELSE ATTRIB/ACK_ERROR_FLAG is set then null


PROCESS NAME:          CHECK FOR ATTRIBUTES ERRORS
PROCESS NUMBER:        1.4.9
PROCESS DESCRIPTION:
IF Input = RECEIVED_ATTRIBUTES_PACKET and no errors exist then
     Output VALID_RECEIVED_ATTRIBUTES_PACKET
ELSEIF Input = RECEIVED_ATTRIBUTES_PACKET and errors exist then
     Output RECEIVED_ATTRIB_ERRORS
ELSEIF Input = RECEIVED_ATTRIB/ACK_PACKET and errors exist then
     Output ATTRIB/ACK_ERROR flag
     Output RECEIVED_ATTRIB_ERRORS
ELSEIF Input = RECEIVED_ATTRIB/ACC_PACKET and errors exist then
     Output ATTRIB/ACC_ERROR flag
     Output RECEIVED_ATTRIB_ERRORS
ELSE Input = RECEIVED_ATTRIB/ACK_PACKET or RECEIVED_ATTRIB/ACC_PACKET and
     no errors exist then null


PROCESS NAME:          CHECK FOR ACCESS ERRORS
PROCESS NUMBER:        1.4.10
PROCESS DESCRIPTION:
IF Input = RECEIVED_ACCESS_PACKET with no error then
     Output VALID_RECEIVED_ACCESS_PACKET
ELSEIF Input = RECEIVED_ACCES_PACKET with errors then
     Output RECEIVED_ACC_ERRORS
ELSEIF Input = RECEIVED_ATTRIB/ACC_PACKET with no errors and ATTRIB/ACC_ERROR
     flag is not set then
          Output VALID_RECEIVED_ACCESS_PACKET
ELSEIF Input = RECEIVED_ATTRIB/ACC_PACKET with errors and the ATTRIB/ACC_ERROR
     flag is not set then
          Output RECEIVED_ACC_ERRORS
```

```
ELSE Input = ATTRIB/ACC_ERROR then null


PROCESS NAME:          DECODE ATTRIBUTES FIELDS
PROCESS NUMBER:        1.4.11
PROCESS DESCRIPTION:
IF Input = VALID_RECEIVED_ATTRIBUTES_PACKET and FIRST_ATTRIB_FLAG is not set
then
     Output SECOND_ATTRIB_FLAG
ELSE Input = VALID_RECEIVED_ATTRIBUTES_PACKET and FIRST_ATTRIB_FLAG is set
then
     Output START_ACCESS
     Reset FIRST-ATTRIB_FLAG


PROCESS NAME:          CODE SETUP ERRORS
PROCESS NUMBER:        1.4.12
PROCESS DESCRIPTION:
IF Input = RECEIVED_CONFIG_ERRORS, RECEIVED_ACK_ERRORS,
           RECEIVED_ATTRIB_ERRORS, RECEIVED_ACC_ERRORS, or RECEIVED_ACCESS_
           COMPLETE_ERRORS then
     Output as SETUP_ERRORS
ELSE Null


PROCESS NAME:          CODE SETUP PACKETS
PROCESS NUMBER:        1.4.13
PROCESS DESCRIPTION:
IF Input = ACCESS_COMPLETE_PACKET, ATTRIB/ACK_PACKET, ACKNOWLEDGE_PACKET,
           ACCESS_PACKET, ACC/ER_PACKET, CONFIGURATION_PACKET, ACC/ECF/SCF_
           PACKET, ATTRIB/ACC_PACKET, or ATTRIBUTES_PACKET then
     Output as SETUP_PACKETS
ELSE Null


PROCESS NAME:          GENERATE ACCESS PACKET
PROCESS NUMBER:        1.5.1
PROCESS DESCRIPTION:
IF Input = RECEIVED_TIMEOUT, RECEIVED_UPSUPPORT_ACCESS, RECEIVED_SYNC_ACCESS,
           or START_ACCESS then
     Output ACCESS_PACKET
ELSE Null


PROCESS NAME:          DECODE ACCESS FIELDS
```

```
PROCESS NUMBER:       1.5.2
PROCESS DESCRIPTION:
IF Input ACCFUNC value = open or create AND ATTRIB/ACC_PACKET was received
then
     Output START_ATTRIB/ACK flag
ELSEIF Input ACCFUNC value = open or create and ATTRIB/ACC_PACKET  was not
     received then
          Output START_ACK flag
ELSEIF Input ACCFUNC value = rename, erase, or execute command file then
     Output START_ER/ECF_COMP flag
ELSEIF Input ACCFUNC value = directory-list then
     Output START_ACC/COMP flag
ELSE Input ACCFUNC value = submit command file then
     Output SUBMIT_AS_COMMAND_FILE


PROCESS NAME:         GENERATE ACCESS COMPLETE PACKET
PROCESS NUMBER:       1.5.3
PROCESS DESCRIPTION:
IF Input = START_ER/ECF_COMP, START_ACC/COMP, RECEIVED_ACCOMP(COMMAND),
           RECEIVED_ACCOMP(PURGE), or RECEIVED_ACCOMP(PURGE) in conjunction
           with a RECEIVED_CONTINUE_ABORT_PACKET then
     Generate and Output an ACCESS_COMPLETE_PACKET(RESPONSE)
ELSEIF Input = ACCOMP(EOS) then
     Output an ACCESS_COMPLETE_PACKET(EOS)
ELSEIF Input = ACCOMP(COMMAND) or RECEIVED_EOF then
     Output an ACCESS_COMPLETE_PACKET(COMMAND)
ELSEIF Input = ACCOMP(PURGE) then
     Output an ACCESS_COMPLETE_PACKET(PURGE)
ELSE Input = RECEIVED_FORMAT_ACC/COMP, RECEIVED_UNSUPPORT_ACC/COMP, RECEIVED_
             SYNC_ACC/COMP, or RECEIVED_TIMEOUT then
     Output the appropriate  ACCESS_COMPLETE_PACKET


PROCESS NAME:         GENERATE ACKNOWLEDGE PACKET
PROCESS NUMBER:       1.5.4
PROCESS DESCRIPTION:
IF Input = RECEIVED_TIMEOUT, START_CONNECT_ACK, RECEIVED_UNSUPPORT_ACK,
           RECEIVED_SYNC_ACK, RECEIVED_FORMAT_ACK, or START_ACK then
     Output an ACKNOWLEDGE_PACKET
ELSE Null


PROCESS NAME:         CHECK FOR ACCESS COMPLETE ERRORS
PROCESS NUMBER:       1.5.5
PROCESS DESCRIPTION:
IF Input = RECEIVED_ACCESS_COMPLETE_PACKET with no errors then
     Output a VALID_RECEIVED_ACCESS_COMPLETE_PACKET
```

```
ELSE Input = RECEIVED_ACCESS_COMPLETE_PACKET with errors then
     Output a RECEIVED_ACCESS_COMPLETE_ERRORS


PROCESS NAME:          DECODE CMPFUNC FIELDS
PROCESS NUMBER:        1.5.6
PROCESS DESCRIPTION:
IF Input CMPFUMC value = terminate then
     Output RECEIVED_ACCOMP(COMMAND)
ELSEIF Input CMPFUNC value = response then
     Output RECEIVED_ACCOMP(RESPONSE)
ELSEIF Input CMPFUNC value = end of stream then
     Output RECEIVED_ACCOMP(EOS)
ELSE Output RECEIVED_ACCOMP(PURGE)


PROCESS NAME:          GENERATE ATTRIB/ACK PACKET
PROCESS NUMBER:        1.5.7
PROCESS DESCRIPTION:
IF Input = RECEIVED_TIMEOUT or START_ATTRIB/ACK then
     Output a ATTRIB/ACK_PACKET
ELSE Null


PROCESS NAME:          DECODE CONTROL PACKET
PROCESS NUMBER:        1.6.1
PROCESS DESCRIPTION:
IF Input = RECEIVED_CONTROL_PACKET with errors then
     Output RECEIVED_CONTROL_ERRORS
ELSEIF Input CTLFUNC value = connect then
     Output RECEIVED_CONNECT_PACKET
ELSEIF Input CTLFUNC value = get then
     Output RECEIVED_GET_PACKET
ELSE Input CTLFUNC value = put, delete, update, or rewing then
     Output RECEIVED_PUT_PACKET


PROCESS NAME:          DECODE CONNECT FIELDS
PROCESS NUMBER:        1.6.2
PROCESS DESCRIPTION:
IF Input + RECEIVED_CONNECT_PACKET then
     Decode the fields and
          Output a START_CONNECT_ACK flag
ELSE Null
```

```
PROCESS NAME:          DECODE GET FIELDS
PROCESS NUMBER:        1.6.3
PROCESS DESCRIPTION:
IF Input RAC field = sequential file access or sequential record access or
                     block mode file transfer then
     Output RECEIVED_SEQ_GET_PACKET
ELSEIF Input RAC field = keyed access then
     Output RECEIVED_KEY_GET_PACKET
ELSE Input RAC field = access by record file address then
     Output RECEIVED_ADD_GET_PACKET


PROCESS NAME:          DECODE PUT FIELDS
PROCESS NUMBER:        1.6.4
PROCESS DESCRIPTION:
IF Input RAC field = sequential file access or sequential record access or
                     block mode file transfer then
     Output RECEIVED_SEQ_PUT_PACKET
ELSEIF Input RAC field = keyed access then
     Output RECEIVED_KEY_PUT_PACKET
ELSE Input RAC field = access by record file address then
     Output RECEIVED_ADD_PUT_PACKET


PROCESS NAME:          DECODE ON NEXT ACTION
PROCESS NUMBER:        1.7.1
PROCESS DESCRIPTION:
IF Input = RECEIVED_SEQ_GET_PACKET or RECEIVED_KEY_GET_PACKET then
     Send out the data in the correct format
ELSEIF Input = RECEIVED_ADD_GET_PACKET then
     Send out the data one record at a time between reception of the RECEIVED_
     SUCCESSFUL_OPERATION status packet
ELSEIF Input = RECEIVED_CONTINUE_ONLY_PACKET then either
     Resend the bad data record and continue or just
     Continue sending data records.
ELSEIF Input = RECEIVED_CONTINUE_SKIP_PACKET then
     Skip over the bad record and continue sending data records
ELSEIF Input = errors in any of the above inputs then
     Output FILE_ERRORS
ELSEIF Input = RECEIVED_SYNC_DATA, RECEIVED_FILE_ERRORS, RECEIVED_FORMAT_
               DATA, RECEIVED_UNSUPPORT_DATA then
                       IF want to skip the bad record and continue then
                            Output SKIP_REC/CONTINUE
                       ELSEIF want to continue with the bad record still in
                            the file then
                                 Output CONTINUE_WITH_BAD_REC
                       ELSEIF want to purge the new file and terminate then
                            Output ACCOMP(PURGE) immediately followed by
                            Output PURGE/ABORT_INTERRUPT
```

```
                    ELSE want to close the new file and terminate then
                         Output ACCOMP(COMMAND) immediately followed by
                         Output PRUGE/ABORT_INTERRUPT
ELSEIF want to stop a sequential file storage operation before it is complete
     and purge the incomplete file then
          Output PURGE/ABORT_INTERRUPT
ELSEIF want to send a message to let the accessing system know there will be a
     delay in getting data then
          Output OPERATION_IN_PROGRESS
ELSE end-of-file is detected and want to end the data stream but not the
     logical link then
          Output ACCOMP(EOS)



PROCESS NAME:          DECODE ON REQUIRED ACTION
PROCESS NUMBER:        1.7.2
PROCESS DESCRIPTION:
IF Input = RECEIVED_DATA_FILE with errors or RECEIVED_FILE_ERRORS then
     Output either DATA_ERRORS or FILE_ERRORS or
          IF want to request link termination then
               Output ACCOMP(COMMAND)
          ELSEIF want to request data stream termination then
               Output ACCOMP(EOS)
          ELSEIF want to request the information be sent again then
               Output RESENT_INFO
          ELSE want to ship that record and continue then
               Output SKIP_INFO/CONTINUE
ELSEIF Input = RECEIVED_DATA_FILE and RECEIVED_SEQ_PUT_PACKET or RECEIVED_
               KEY_PUT_PACKET or SUBMIT_AS_COMMAND_FILE then
     Output data to file in correct format
ELSEIF Input = RECEIVED_DATA_FILE and RECEIVED_ADD_PUT_PACKET then
     Output data to file in correct format and after each store operation
     Output SUCCESSFUL_OPERATION
ELSE Input = RECEIVED_OPERATION_IN_PROGERSS then do not
     Output FILE_ERRORS or DATA_ERRORS



PROCESS NAME:          GENERATE CONTINUE ABORT PACKET
PROCESS NUMBER:        1.7.3
PROCESS DESCRIPTION:
IF Input = PURGE/ABORT_INTERRUPT or CONTINUE_ABORT_ERROR then
     Output CONTINUE_ABORT_PACKET
ELSE Null



PROCESS NAME:          GENERATE CONTINUE SKIP PACKET
PROCESS NUMBER:        1.7.4
PROCESS DESCRIPTION:
```

```
IF Input = SKIP_REC/CONTINUE, SKIP_INFO/CONTINUE, or CONTINUE_SKIP_ERROR then
     Output CONTINUE_SKIP_PACKET
ELSE Null


PROCESS NAME:          GENERATE CONTINUE ONLY PACKET
PROCESS NUMBER:        1.7.5
PROCESS DESCRIPTION:
IF Input = CONTINUE_WITH_BAD_REC, RESEND_INFO, or CONTINUE_ONLY_ERROR then
     Output CONTINUE_ONLY_PACKET
ELSE Null


PROCESS NAME:          CODE CONTINUE PACKETS
PROCESS NUMBER:        1.7.6
PROCESS DESCRIPTION:
IF Input = CONTINUE_ABORT_PACKET, CONTINUE_SKIP_PACKET, or CONTINUE_ONLY_
           PACKET then
     Output as CONTINUE_INTERRUPT_PACKET
ELSE Null


PROCESS NAME:          GENERATE DATA PACKET
PROCESS NUMBER:        1.7.7
PROCESS DESCRIPTION:
IF Input = DATA then
     Generate and output a DATA_PACKET
ELSE Input = DATA + EOF then
     Generate and output last DATA_PACKET and
     Output EOF flag


PROCESS NAME:          CODE CONTINUE ERRORS
PROCESS NUMBER:        1.7.8
PROCESS DESCRIPTION:
IF Input = RECEIVED_UNSUPPORT_CONTINUE, RECEIVED_FORMAT_CONTINUE, or RECEIVED_
           SYNC_CONTINUE then
     IF CONTINUE_ABORT_PACKET was the last continue packet sent then
          Output CONTINUE_ABORT_ERROR
     ELSEIF CONTINUE_SKIP_PACKET was the last continue packet sent then
          Output CONTINUE_SKIP_ERROR
     ELSE CONTINUE_ONLY_PACKET was the last continue packet sent then
          Output CONTINUE_ONLY_ERROR
ELSE Null
```

```
PROCESS NAME:          START/STOP TIMER
PROCESS NUMBER:        1.8.1
PROCESS DESCRIPTION:
IF Input = RECEIVED_CONFIGURATION_PACKET, RECEIVED_ATTRIBUTES_PACKET,
           RECEIVED_ACCESS_PACKET, RECEIVED_ATTRIB/ACC_PACKET, ACCESS_
           COMPLETE_PACKET(RESPONSE), or SETUP_ERRORS then
    START TIMER
ELSEIF Input = CONFIGURATION_PACKET, ATTRIBUTES_PACKET, ACKNOWLEDGE_PACKET,
               RECEIVED_CONFIGURATION_PACKET, LINK_TERMINATION, or RECEIVED_
               STARTUP_PACKETS then
    STOP TIMER
ELSE output TIMEOUT


PROCESS NAME:          CODE WORKING PACKETS
PROCESS NUMBER:        1.8.2
PROCESS DESCRIPTION:
IF Input = SETUP_PACKETS, DATA_PACKETS, CONTROL_PACKETS, or CONTINUE_
           INTERRUPT_PACKETS then
     Output WORKING_PACKETS
ELSE Null


PROCESS NAME:          GENERATE STATUS PACKETS
PROCESS NUMBER:        1.8.3
PROCESS DESCRIPTION:
IF Input = RECEIVED_CONTROL_ERRORS, RECEIVED_UNSUPPORT_STATUS2, RECEIVED_
           FORMAT_STATUS2, RECEIVED_SYNC_STATUS2, SETUP_ERRORS, EOF, FILE_
           ERRORS, SUCCESSFUL_OPERATION, OPERATION_IN_PROGRESS, RECEIVED_
           STATUS_ERROR, DATA_ERRORS, or CONTINUE_ERRORS then
     Output  appropriate status packet as STATUS_PACKET
ELSE Null


PROCESS NAME:          TERMINATE LOGICAL LINK
PROCESS NUMBER:        1.8.4
PROCESS DESCRIPTION:
IF Input = TIMEOUT, RECEIVED_UNSUPPORT_MISC, RECEIVED_FORMAT_MISC, RECEIVED_
           SYNC_UNKNOWN, or DISCONNECT then
     Output DISCONNECT_REQUEST
ELSE Null


PROCESS NAME:          TERMINATE DATA STREAM
PROCESS NUMBER:        1.8.5
PROCESS DESCRIPTION:
IF Input = RECEIVED_ACCOMP(EOS) then
```

```
     IF want to start new access then
          Output NEW_ACCESS
     ELSE output DISCONNECT
ELSE Input = RECEIVED_ACCOMP(RESPONSE) then
     Output DISCONNECT
```

```
PROCESS NAME:          CODE FILE PACKETS
PROCESS NUMBER:        1.8.6
PROCESS DESCRIPTION:
IF Input = DISCONNECT_REQUEST, WORKING_PACKETS, or STATUS_PACKETS then
     Output as OUTGOING_PRIMARY_NODE_ARCHITECTURE_PACKET
ELSE Null
```

```
PROCESS NAME:          CHECK FOR CONTINUE ERROR
PROCESS NUMBER:        1.9
PROCESS DESCRIPTION:
IF Input = CONTINUE_PACKETS with errors then
     Output CONTINUE_ERRORS
ELSEIF Input CONFUNC value = try again then
     Output as RECEIVED_CONTINUE_ONLY_PACKET
ELSEIF Input CONFUNC value = skip then
     Output as RECEIVED_CONTINUE_SKIP_PACKET
ELSE Input CONFUNC value = abort then
     Output as RECEIVED_CONTINUE_ABORT_PACKET
```

```
PROCESS NAME:          EXECUTE ARCHITECETURE PROTOCOL AT SECONDARY NODE
PROCESS NUMBER:        2.0
PROCESS DESCRIPTION:
     Provide standardized formats and procedures for accessing and passing
     data between a user process and a file system existing in a network
     enviroment.
```

DATA DICTIONARY

FOR TRANSPORT/NETWORK LEVEL (T/N) PROTOCOL

Contents

DATA ELEMENT NAME: ABORT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE RECEIVED_DISCONNECT_ INITIATE_PACKET IS A RESULT OF AN ABORT_COMMAND. RESULTS IN THE IMMEDIATE GENERATION OF A DISCONNECT_CONFIRM_ PACKET.
NOTES: EXECUTE DISCONNECT PACKET LAYER

DATA ELEMENT NAME: ABORT_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
OPERATOR OR DIALOGUE COMMAND USED TO ABORT A LOGICAL LINK CONNECTION, DIALOGUE DATA IS NOT SAVED, CURRENT DIALOGUE DATA BEING TRANSMITTED IS LOST.
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: ACCEPT_CONFIRM
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO SET THE LINK_ACCESSABLE BIT IN THE ADJACENT_NODE_PARAMETERS TABLE. THIS INDICATES THAT THE LOGICAL_LINK HAS BEEN SUCCESSFUL CONNECTED.
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: ACCEPT_CONNECT
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO INDICATE THE ACCEPTANCE OF A RECEIVED_ CONNECT_INITIATE_PACKET. RESULTS IN THE GENERATION OF A CONNECT_CONFIRM_PACKET.
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: ACCOUNT
ALIASES: NONE
VALUES AND MEANINGS:
A CHARACTER-CODED DEFINITION THAT WHEN PAIRED WITH THE REQUESTOR-ID IDENTIFIES A "BILLING ADDRESS" FOR SERVICE COSTS AT THE DESTINATION NODE.

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: ACKNUM
ALIASES: NONE
VALUES AND MEANINGS:

THE NUMBER OF THE LAST TRANSPORT DATA SEGMENT SUCCESSFULLY
RECEIVED AND AN ACK OR NAK INDICATION. THIS FIELD IS OPTIONAL. ITS PRESENCE IS INDICATED BY BIT 15 BEING SET.
FORMAT:

QUAL = ACK = ACKNOWLEDGE
NAK = NEGATIVE ACKNOWLEDGE
NUMBER = THE SEGMENT NUMBER

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: ACKNUMI
ALIASES: NONE
VALUES AND MEANINGS:

THE NUMBER OF THE LAST TRANSPORT INTERRUPT OR LINK SERVICE PACKET SUCCESSFULLY RECEIVED AND AN ACK OR NAK INDICATION. THIS FIELD IS OPTIONAL. ITS PRESENCE IS INDICATED BY BIT 15 BEING SET.
FORMAT:

QUAL = ACK = ACKNOWLEDGE
NAK = NEGATIVE ACKNOWLEDGE
NUMBER = THE PACKET NUMBER

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: ADJACENT_NODE_PACKETS
ALIASES: NONE
COMPOSITION:

ADJACENT_NODE_PACKETS =

| TRANSMITTED_TRANSPORT_INITIALIZATION_PACKET |
| FIRST_DISCONNECT_CONFIRM_PACKET |
| TRANSMITTED_TRANSPORT_DATA_PACKET |
| TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET |
| TRANSMITTED_TRANSPORT_CONTROL_PACKET (EXCEPT FOR CONNECT_INITIATE) |

= WHERE DSTADDR IMPLYS THAT THE PACKET IS GOING TO AN ADJACENT NODE. INITIALIZATION PACKETS ALWAYS ARE BETWEEN ADJACENT NODES ONLY.

NOTES: EXECUTE OUTGOING TRANSPORT PACKET LAYER

DATA FLOW NAME: ADJACENT_NODE_ROUTING_PACKET
ALIASES: ROUTING_PACKET
VALID_ROUTING_PACKET
HOPPED_ROUTING_PACKET
INITIAL_ROUTE_PACKET
OLD_ROUTING_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: BLKSIZE
ALIASES: NONE
VALUES AND MEANINGS:
THE MAXIMUM PHYSICAL BLOCK SIZE THE LINK WILL ACCEPT.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: CHOKE
ALIASES: NONE
VALUES AND MEANINGS:
BIT INDICATES THAT THE PACKET IS A CHOKE PACKET RATHER THAN AN ORDINARY DATA PACKET. NOT PRESENTLY USED.
NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: CLOSED_FLOW_CONTROL
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG THAT INDICATES THAT THE FLOW_CONTROL_PARAMETER DATA_FLOW_CONTROL_SWITCH IS SET TO CLOSED RESULTING IN ALL INCOMING DATA PACKETS BEING NEGATIVELY ACKNOWLEDGED.
NOTES: EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: COMMVER
ALIASES: NONE
VALUES AND MEANINGS:
THE VERSION OF THE COMMUNICATIONS PART OF THE TRANSPORT PROTOCOL.
NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: CONNECT_CONFIRM_PACKET
ALIASES: RECEIVEL_CONNECT_CONFIRM_PACKET
COMPOSITION:

CONNECT_CONFIRM_PACKET = MSGFLG + DSTADDR + SRCADDR +
SEGMENT REQUESTS COUNTS + INFO +
SEGSIZE + DSTNAME + SRCNAME +
MENU + RQSTRID + PASSWRD +
ACCOUNT

NOTES: EXECUTE CONNECT PACKET LAYER

DATA FLOW NAME: CONNECT_INITIATE_PACKET
ALIASES: RECEIVED_CONNECT_INITIATE_PACKET
COMPOSITION:

CONNECT_INITIATE_PACKET = MSGFLG + DSTADDR=0 + SRCADDR +
SEGMENT REQUEST COUNT + INFO +
SEGSIZE + DSTNAME + SRCNAME +
MENU + RQSTRID + PASSWRD +
ACCOUNT

NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: CONNECT_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:

OPERATOR OR DIALOGUE COMMAND TO START CONNECTION OF THE
LOGICAL LINK.

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: CORRECT_PASSWORD
ALIASES: NONE
VALUES AND MEANINGS:

THE RECEIVED CORRECT PASSWORD

NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: CORRECT_PASSWORD_COMMAND
ALIASES: RECEIVED_CORRECT_PASSWORD_COMMAND
VALUES AND MEANINGS:

A COMMAND TO NOTIFY THE SATELLITE NODE THAT THE PASSWORD
GIVEN IN THE NODE_VERIFICATION_PACKET WAS CORRECT.

LOGICAL LINK ESTABLISHMENT CAN NOW PROCEED. ADJACENT_NODE_PARAMETER TABLE VALUES ARE NOW VALID.

NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: COUNTED_TRANSPORT_DATA_SEGMENT

ALIASES:
INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: COUNTED_TRANSPORT_I/L_PACKET

ALIASES:
TRANSPORT_I/L_PACKET
RETRANSMITTED_I/L_PACKET
TRANSMITTED_I/L_PACKET

COMPOSITION:

```
COUNTED_TRANSPORT_I/L_PACKET =
                              --                                --
                              | INTERRUPT_LINK_SERVICES_PACKET |
                              | DATA_LINK_SERVICES_PACKET      |
                              --                                --
```

NOTES: EXECUTE I/L PACKET LAYER

DATA ELEMENT NAME: DATA

ALIASES: NONE

VALUES AND MEANINGS:
THE DATA THE DIALOGUE PROCESS WISHES TO SEND OVER A LOGICAL LINK. THIS INFORMATION WILL BE TOTALLY TRANSPARENT AND MAY BE ALL 8-BITS OF EACH BYTE. DATA PACKETS ARE LIMITED TO THE MAXIMUM SEGSIZE ALLOWED ON THE LOGICAL LINK IN THE DIRECTION THAT THE PACKET IS SENT. THE LENGTH OF THE DATA FIELD IS ASCERTAINED FROM THE TOTAL LENGTH OF THE NORMAL DATA SEGMENT AND CONSIST OF ALL BYTES IN THE SEGMENT AFTER THE SEGNUM FIELD. THE DATA FIELD MAY BE NULL.

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: DATA_ACK_FLAG
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO START THE GENERATION OF AN ACKNOWLEDGEMENT PACKET ON A RECIEVED DATA PACKET.
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: DATA_ACK_PACKET
ALIASES: DATA_NAK_PACKET
COMPOSITION:
DATA_ACK_PACKET = MSGFLG + DSTADDR + SRCADDR + ACKNUM
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: DATA_FLOW_ERROR
ALIASES: NONE
VALUES AND MEANINGS:
INDICATES THAT THE FLOW_CONTROL_PARAMETER TABLE CONTAINED A CLOSED DATA_FLOW_CONTROL_SWITCH OR THE DATA_REQUEST_COUNT HAD BEEN OVERFLOWED.
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA ELEMENT NAME: DATAI
ALIASES: NONE
VALUES AND MEANINGS:
THE DATA TO BE SENT OVER A LOGICAL LINK. THIS FIELD IS TOTALLY TRANSPARENT AND MAY USE ALL 8-BITS OF EACH BYTE. THE LENGTH OF THE DATA FIELD IS ASCERTAINED FROM THE TOTAL LENGTH OF THE INTERRUPT PACKET AND CONSIST OF ALL BYTES IN THE PACKET AFTER THE SEGNUMI FIELD.
NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: DATA_LINK_SERVICES_PACKET
ALIASES: INTERRUPT_LINK_SERVICES_PACKET
VALID_LINK_SERVICES_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE I/L PACKET LAYER

DATA ELEMENT NAME: DATA_NAK_FLAG
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO START THE GENERATION OF AN NEGATIVE ACKNOWLEDGEMENT PACKET ON A RECEIVED DATA PACKET.
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: DATA_NAK_PACKET
ALIASES: DATA_ACK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: *EXECUTE DIALOGUE SEGMENT LAYER*
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: DESTINATION
ALIASES: NONE
VALUES AND MEANINGS:
THE NAME OF THE NODE TO WHICH THE ROUTING PACKET IS GOING.
NOTES: ALL NETWORK LAYERS

DATA FLOW NAME: DISCONNECT_CONFIRM_PACKET
ALIASES: FIRST_DISCONNECT_CONFIRM_PACKET
RECEIVED_DISCONNECT_CONFIRM_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER

DATA FLOW NAME: DISCONNECT_INITIATE_PACKET
ALIASES: RECEIVED_DISCONNECT_INITIATE_PACKET
COMPOSITION:
DISCONNECT_INITIATE_PACKET = MSGFLG + DSTADDR + SRCADDR + DISCONNECT REASON + DSTNAME + SRCNAME + MENU + RQSTRID + PASSWRD + ACCOUNT
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: DISCONNECT_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:

OPERATOR OR DIALOGUE COMMAND USED TO STOP THE LOGICAL LINK PROCESS. CURRENT DIALOGUE DATA IS TRANSMITTED AND ACKNOWLEDGED BEFORE THE DISCONNECT_REQUEST IS ALLOWED TO TAKE EFFECT.

NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: DISCONNECT_REQUIRED
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THAT THE SATELLITE_NODE_PARAMETERS TABLE IS NOT SET TO THE CORRECT PASSWORDS AND THAT THE LINK IS NOT UP AND RUNNING.

NOTES: DECODE ROUTE HEADER LAYER

DATA ELEMENT NAME: DSTADDR
ALIASES: NONE
VALUES AND MEANINGS:

THE LOGICAL LINK DESTINATION ADDRESS FOR THE PACKET. THIS ADDRESS IS ASSIGNED WHEN A LINK IS EXTABLISHED (CONNECT_PACKETS).

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: DSTNAME
ALIASES: NONE
VALUES AND MEANINGS:

THE DESTINATION PROCESS IDENTIFICATION.
FORMAT 02:
OBJTYPE = OBJECT TYPE
GRPCODE = BINARY GROUP CODE
USRCODE = BINARY USER CODE
DESCRPT = PROCESS DESCRIPTOR. A UNIQUE NAME THAT QUALIFIES THE OBJECT TYPE.
FORMAT 01:
OBJTYPE = OBJECT TYPE
DESCRPT = PROCESS DESCRIPTOR

```
                        FORMAT 00:
                              OBJTYPE = OBJECT TYPE
                        RULES:
                              APPLICATION PROGRAMS = FORMAT 01 OR 02
                              DIALOGUE LEVEL PROTOCOL = FORMAT 00
                              SOURCE PROCESS DESCRIPTOR = FORMAT 02
NOTES:                  ALL TRANSPORT LAYERS


DATA ELEMENT NAME:      ENTER_MAINTENANCE_MODE
ALIASES:                NONE
VALUES AND MEANINGS:
                        REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL
NOTES:                  EXECUTE DIALOGUE SEGMENT LAYER


DATA ELEMENT NAME:      ERROR_REASON
ALIASES:                NONE
VALUES AND MEANINGS:
                        REFER TO FILE "REASON"
NOTES:                  EXECUTE DISCONNECT PACKET LAYER
                        EXECUTE CONTROL PACKET LAYER
                        EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER


DATA ELEMENT NAME:      FCVAL
ALIASES:                NONE
VALUES AND MEANINGS:
                        THE NUMBER OF NORMAL DATA SEGMENTS OR INTERRUPT PACKETS
                        THAT THE SENDER OF THE PACKET CAN RECEIVE IN ADDITION TO
                        THOSE PREVIOUSLY REQUESTED BY A LINK SERVICE PACKET. THIS
                        NUMBER IS ADDED TO THE REQUEST COUNT WHICH IS MAINTAINED
                        IN THE FLOW_CONTROL_PARAMETERS TABLE TO DETERMINE HOW
                        MANY NORMAL DATA SEGMENTS OR INTERRUPT PACKETS WILL BE
                        TRANSMITTED VIA A LOGICAL LINK.
NOTES:                  ALL TRANSPORT LAYERS


DATA FLOW NAME:         FIRST_DISCONNECT_CONFIRM_PACKET
ALIASES:                RECEIVED_DISCONNECT_CONFIRM_PACKET
                        DISCONNECT_CONFIRM_PACKET
COMPOSITION:
                        FIRST_DISCONNECT_CONFIRM_PACKET = MSGFLG + DSTADDR +
                                                          SRCADDR + REASON
NOTES:                  DECODE ROUTE HEADER LAYER
                        EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
                        EXECUTE OUTGOING TRANSPORT PACKETS LAYER
```

DATA FLOW NAME: FLOW_CONTROL_ERRORS
ALIASES: NONE
COMPOSITION:

```
                                 __                    __
FLOW_CONTROL_ERRORS = | LINK_INACCESSABLE     |
                      | INTERRUPT_FLOW_ERROR  |
                      | DATA_FLOW_ERROR       |
                      --                      --
```

NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: FUNCTIONS
ALIASES: NONE
VALUES AND MEANINGS:

THE FUNCTIONS SUPPORTED AT THIS NODE.
FORMAT:

```
      INT = INTERCEPT FUNCTIONS
          = 000 = NO INTERCEPT - SET ON TRANSMIT BY A
                                 NON-INTERCEPT-SATELLITE
                                 NODE
          = 111 = INTERCEPT    - SET ON TRANSMIT BY A
                                 INTERCEPT-ROUTING-NODE
```

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: HOP_COUNT
ALIASES: HOPS
VALUES AND MEANINGS:

REFER TO ALIASES
NOTES: ALL NETWORK LAYERS

DATA FLOW NAME: HOPPED_ROUTING_PACKET
ALIASES: ROUTING_PACKET
VALID_ROUTING_PACKET
INITIAL_ROUTING_PACKET
ADAJACENT_NODE_ROUTING_PACKET
OLD_ROUTING_PACKETS
COMPOSITION:

SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: HOPS
ALIASES: HOP_COUNT

VALUES AND MEANINGS:
COUNTER USED TO KILL OFF OLD PACKETS.
NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: I/L_ACK_FLAG
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO START THE GENERATION OF AN ACKNOWLEDGEMENT PACKET ON A RECEIVED_I/L_PACKET.
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: I/L_ACK_PACKET
ALIASES: I/L_NAK_PACKET
COMPOSITION:
I/L_ACK_PACKET = MSGFLG + DSTADDR + SRCADDR + ACKNUMI
NOTES: EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: I/L_NAK_FLAG
ALIASES: *NONE*
VALUES AND MEANINGS:
FLAG USED TO START THE GENERATION OF AN NEGATIVE ACKNOWLEDGEMENT PACKET ON A RECEIVED I/L PACKET.
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: I/L_NAK_PACKET
ALIASES: I/L_ACK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER

DATA FLOW NAME: INCOMING_DIALOGUE_DATA
ALIASES: NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT

TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET

COMPOSITION:

INCOMING_DIALOGUE_DATA = MSGFLG + DSTADDR + SRCADDR +
ACKNUM + SEGNUM + DATA

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: INCOMING_DIALOGUE_INTERRUPT
ALIASES: TRANSPORT_INTERRUPT_PACKET
PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET
VALID_INTERRUPT_PACKET

COMPOSITION:

INCOMING_DIALOGUE_INTERRUPT = MSGFLG + DSTADDR + SRCADDR
+ACKNUMI + SEGNUMI + DATAI

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA FLOW NAME: INCOMING_DIALOGUE_MESSAGE
ALIASES: SECONDARY_INCOMING_DIALOGUE_MESSAGE
COMPOSITION:

INCOMING_DIALOGUE_MESSAGE =

INCOMING_DIALOGUE_PACKET
OPERATOR_PASSWORD_COMMAND
CONNECT_REQUEST
ENTER_MAINTENANCE_NODE
OPERATOR_START_COMMAND
ABORT_COMMAND
DISCONNECT_REQUEST

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA FLOW NAME: INCOMING_DIALOGUE_PACKET
ALIASES: OUTGOING_DIALOGUE_DATA_PACKET
COMPOSITION:

INCOMING_DIALOGUE_PACKET =

INCOMING_DIALOGUE_INTERRUPT
INCOMING_DIALOGUE_DATA

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: INCOMING_DIALOGUE_SEGMENT
ALIASES: INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: INCOMING_NODE_NETWORK_PACKET
ALIASES: SECONDARY_INCOMING_NODE_NETWORK_PACKET
COMPOSITION:

$$\text{INCOMING\_NODE\_NETWORK\_PACKET} = \left[\begin{array}{l} \text{NETWORK\_PACKET} \\ \text{ROUTING\_PACKET} \end{array}\right]$$

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA FLOW NAME: INCOMING_NODE_TRANSPORT_PACKET
ALIASES: INCOMING_SATELLITE_TRANSPORT_PACKET
SECONDARY_INCOMING_SATELLITE_TRANSPORT_PACKET
SECONDARY_INCOMING_NODE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
DECODE ROUTE HEADER LAYER
EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: INCOMING_SATELLITE_TRANSPORT_PACKET
ALIASES: SECONDARY_INCOMING_SATELLITE_TRANSPORT_PACKET
INCOMING_NODE_TRANSPORT_PACKET
SECONDARY_INCOMING_NODE_TRANSPORT_PACKET
COMPOSITION:
INCOMING_SATELLITE_TRANSPORT_PACKET =

| OUTGOING_TRANSPORT_ROUTE_PACKET |
| RECIEVED_ADJACENT_NODE_PACKETS |
| VALID_DSTNODE_CI_PACKET |
| INVALID_DSTNODE_PACKETS |
| VALID_DSTNODE_PACKETS |

NOTES: OVERVIEW LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
DECODE ROUTE HEADER LAYER

DATA ELEMENT NAME: INCORRECT_PASSWORD_COMMAND
ALIASES: RECEIVED_INCORRECT_PASSWORD_COMMAND
VALUES AND MEANINGS:
A COMMAND TO NOTIFY THE SATELLITE NODE THAT THE PASSWORD GIVEN IN THE NODE_VERIFICATION_PACKET WAS INCORRECT. BEFORE LOGICAL LINK ESTABLISHMENT CAN PROCEED MUST START INITIALIZATION PROCEDURE OVER WITH CORRECT PASSWORD.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: INFO
ALIASES: NONE
VALUES AND MEANINGS:
INFORMATION. FORMAT:
PRI = LINK PRIORITY NOT USED AT PRESENT.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: INITIALIZE_LINK
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: INITIALIZATION_COMPLETE
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: INITIZATION_ON_OTHER_END
ALIASES: NONE

VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: INITIAL_ROUTE_PACKET
ALIASES: ROUTING_PACKET
VALID_ROUTING_PACKET
HOPPED_ROUTING_PACKET
ADJACENT_NODE_ROUTING_PACKET
OLD_ROUTING_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: INTERRUPT_ERROR
ALIASES: NONE
VALUES AND MEANINGS:
FLOW CONTROL FAILURE = INTERRUPT MESSAGE RECEIVED WHEN THE INTERRUPT REQUEST COUNT IS ZERO.
NOTES: EXECUTE DATA PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: INTERRUPT_FLOW_ERROR
ALIASES: NONE
VALUES AND MEANINGS:

INDICATES THAT IN THE FLOW_CONTROL_PARAMETERS TABLE THAT THE INTERRUPT_REQUEST_COUNT HAD BEEN OVERFLOWED.
NOTES: EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA FLOW NAME: INTERRUPT_LINK_SERVICES_PACKET
ALIASES: DATA_LINK_SERVICES_PACKET
VALID_LINK_SERVICES_PACKET
COMPOSITION:
INTERRUPT_LINK_SERVICES_PACKET = MSGFLG + DSTADDR +
SRCADDR + ACKNUMI +
SEGNUMI + LSFLAGS +
FCVAL
NOTES: EXECUTE I/L PACKET LAYER

DATA ELEMENT NAME: INVALID_DSTADDR_PACKETS
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE DSTADDR FIELD WITHIN THE MSGFLG FIELD DOES NOT CONTAIN AN ADDRESS OF A REACHABLE LOGICAL LINK FROM THIS NODE.
NOTES: DECODE ROUTE HEADER LAYER

DATA ELEMENT NAME: INVALID_DSTNODE_PACKETS
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO INDICATE THAT THE DESTINATION CODE WITHIN THE RTHDR FIELD IS NOT EQUAL TO THE NAME OF A REACHABLE SATELLITE FROM THIS NODE.
NOTES: DECODE ROUTE HEADER LAYER

DATA ELEMENT NAME: LINE_COST
ALIASES: NONE
VALUES AND MEANINGS:
WEIGHT OF A PARTICULAR PHYSICAL LINK.
NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: LINK_INACCESSABLE
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG THAT INDICATES THAT THE ADJACENT_NODE_PARAMETERS TABLE HAS BEEN SET TO A LINK_INACCESSABLE SINCE THE PHYSICAL LINK OVER WHICH THE ADJACENT NODE INITIALIZATION THAT TOOK PLACE HAS UNDERGONE A TRANSITION OUT OF THE ON STATE.
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA ELEMENT NAME: LINK_SERVICES_ERROR
ALIASES: NONE
VALUES AND MEANINGS:
FLOW CONTROL VIOLATION - ILLEGAL FCVAL
NOTES: EXECUTE DATA PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: LS_CODE
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT A TRANSMIT BUFFER WAS NOT COMPLETELY FULL WHEN THE LAST DATA SEGMENT WAS PUT INTO IT.
NOTES: EXECUTE I/L PACKET LAYER
EXECUTE DATA PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: LSFLAGS
ALIASES: NONE
VALUES AND MEANINGS:
LINK SERVICE FLAGS. FORMAT:
FCMOD = FLOW CONTROL MODIFICATION
= 0 = NO CHANGE
= 1 = STOP DATA
= 2 = START DATA
FCVAL INT = INTERPRETATION OF FCVAL FIELD
= 0 = DATA SEGMENT COUNT
= 1 = INTERRUPT REQUEST
COUNT
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: MAXLNKS
ALIASES: NONE
VALUES AND MEANINGS:
THE MAXIMUM NUMBER OF LINKS THIS NODE WILL SUPPORT. THE VALUE IS LIMITED TO 4096.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: MENU
ALIASES: NONE
VALUES AND MEANINGS:
FIELD FORMAT CONTROL = RQSTRID, PASSWORD, ACCOUNT FIELDS
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: MSGDATA

ALIASES: NONE
VALUES AND MEANINGS:

DATA. THE REMAINDER OF A TRANSPORT PACKET
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: MSGFLG
ALIASES: NONE
VALUES AND MEANINGS:

00000000 = MIDDLE SEGMENT OF A MULTI-SEGMENT DIALOGUE MESSAGE
00100000 = THE FIRST SEGMENT OF A MULTI-SEGMENT DIALOGUE MESSAGE
01000000 = THE LAST SEGMENT OF A MULTI-SEGMENT DIALOGUE MESSAGE
01100000 = THE ONLY SEGMENT OF A DIALOGUE MESSAGE
00110000 = INTERRUPT PACKET
00010000 = LINK SERVICE PACKET
00000100 = ACKNOWLEDGEMENT OF NORMAL DATA SEGMENT
00010100 = ACKNOWLEDGEMENT OF INTERRUPT PACKET OR LINK SERVICES PACKET
00011000 = CONNECT_INITIATE_PACKET
00101000 = CONNECT_CONFIRM_PACKET
00111000 = DISCONNECT_INITIATE_PACKET
01001000 = DISCONNECT_CONFIRM_PACKET
01011000 = NODE_INITIALIZATION_PACKET
01011000 = NODE_VERIFICATION_PACKET
00001000 = NO OPERATION (NOP)
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: NODEADDR
ALIASES: NONE
VALUES AND MEANINGS:

THE SOURCE NODE ADDRESS. THE VALUE OF THIS FIELD MUST BE GREATER THAN 1 AND LESS THAN 241. NO TWO NODES IN THE SAME NETWORK MAY HAVE THE SAME NODE ADDRESS.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: NODENAME
ALIASES: RTHDR(SRCNODE)
VALUES AND MEANINGS:

SEE ALIASES
NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: NETWORK_PACKET
ALIASES: NONE
COMPOSITION:

NETWORK_PACKET = [INCOMING_NODE_TRANSPORT_PACKET | NETWORK_TO_NETWORK_PACKET]

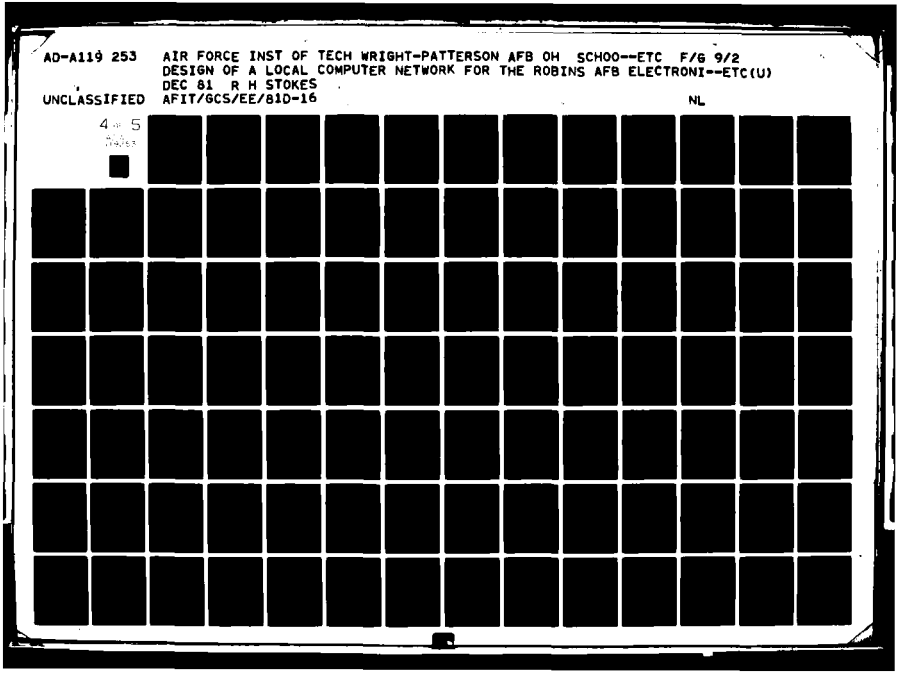
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: NETWORK_TO_NETWORK_PACKET
ALIASES: VALID_NETWORK_TO_NETWORK_PACKET
VALID_NETWORK_PACKET
COMPOSITION:
NETWORK_TO_NETWORK_PACKET = TRANSPORT_PACKET WITH A DESTINATION CODE THAT IS NOT ADJACENT TO THIS NODE.
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: NO_ABORT
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE RECEIVED_DISCONNECT_INITIATE_PACKET IS A NORMALLY GENERATED DISCONNECT. RESULTS IN A STOP_LINK FLAG BEING ISSUED.
NOTES: EXECUTE DISCONNECT PACKET LAYER

DATA ELEMENT NAME: NO_BUFFER_SPACE
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO GENERATE A NEGATIVE ACKNOWLEDGEMENT SINCE THERE WERE NO MORE RECEIVE BUFFERS TO LOAD INCOMING DATA PACKETS INTO.
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: NODE_INITIALIZATION_V0_PACKET
ALIASES: RECEIVED_NODE_INITIALIZATION_PACKET
RECEIVED_INTERCEPT_INITIALIZATION_PACKET
RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET
NODE_INITIALIZATION_V1_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKET LAYER


AD-A119 253
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONI--ETC(U)
DEC 81 R H STOKES
UNCLASSIFIED AFIT/GCS/EE/81D-16
NL
4 OF 5

DATA FLOW NAME: NODE_INITIALIZATION_V1_PACKET
ALIASES: RECEIVED_NODE_INITIALIZATION_PACKET
RECEIVED_INTERCEPT_INITIALIZATION_PACKET
RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET
NODE_INITIALIZATION_V0_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: NODE_VERIFICATION_PACKET
ALIASES: RECEIVED_NODE_VERIFICATION_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: NONADJACENT_NODE_PACKETS
ALIASES: NONE
COMPOSITION:
NONADJACENT_NODE_PACKETS =

| TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET |
| FIRST_DISCONNECT_CONFIRM_PACKET |
| TRANSMITTED_TRANSPORT_CONTROL_PACKET |
| REANSMITTED_TRANSPORT_DATA_PACKET |

= WHERE THE DSTADDR IMPLYS THAT THE PACKET IS GOING TO A NON-ADJACENT NODE. IF A CONNECT_INITIATE_PACKET THEN ALWAYS GIVES THIS ROUTE.

NOTES: EXECUTE OUTGOING TRANSPORT PACKET LAYER

DATA FLOW NAME: NONADJACENT_ROUTE_PACKET
ALIASES: OUTGOING_TRANSPORT_ROUTE_PACKET
OUTGOING_NODE_TRANSPORT_PACKET
SECONDARY_OUTGOING_NODE_TRANSPORT_PACKET
TRANSPORT_PACKET
COMPOSITION:
NONDAJACENT_ROUTE_PACKET = CONTAINS A RTHDR FIELD THAT HAS A DSTNODE NAME THAT IS REFERING TO A SATELLITE THAT IS NOT ADJACENT TO THIS ROUTING NODE.
NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER

EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: NONADJACENT_SATELLITE_PACKETS
ALIASES: NONE
COMPOSITION:

NONADJACENT_SATELLITE_PACKETS = RTHDR_NONADJACENT_NODE_PACKETS - NONADJACENT_ROUTE_PACKET

NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: NORMAL_DATA_PACKET
ALIASES: INCOMING_DIALOGUE_DATA
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET
COMPOSITION:

SEE ALIASES

NOTES: EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: NSPSIZE
ALIASES: NONE
VALUES AND MEANINGS:

THE MAXIMUM TRANSPORT PACKET SEGMENT SIZE THIS NODE WILL ACCEPT. THIS NUMBER MUST BE LESS THAN OR EQUAL TO BLKSIZE

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: OBJTYPE
ALIASES: NONE
VALUES AND MEANINGS:

1. GENERAL TASK, USER PROCESS
2. FILE ACCESS
3. UNIT RECORD SERVICES
4. APPLICATION TERMINAL SERVICES
5. COMMAND TERMINAL SERVICES
6. RSX-11M TASK CONTROL
7. OPERATOR SERVICES INTERFACE
8. NODE RESOURCE MANAGER

NOTES: ALL TRANSPORT LAYERS

DATA FLOW DAME: OLD_ROUTING_PACKETS
ALIASES: ROUTING_PACKET
VALID_ROUTING_PACKET
HOPPED_ROUTING_PACKET
INITIAL_ROUTE_PACKET
ADJACENT_NODE_ROUTING_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: OPERATOR_PASSWORD_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
OPERATOR COMMAND PROVIDED DURNING THE NODE VERIFICATION PHASE OF INITIALIZATION.
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: OPERATOR_START_COMMAND
ALIASES: NONE
VALUES AND MEANINGS:
OPERATOR COMMAND TO START INITIALIZATION OF THE PHYSICAL LINK.
NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA FLOW NAME: OUTGOING_DIALOGUE_DATA_PACKET
ALIASES: INCOMING_DIALOGUE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DATA PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_DIALOGUE_MESSAGE
ALIASES: SECONDARY_OUTGOING_DIALOGUE_MESSAGE
COMPOSITION:

OUTGOING_DIALOGUE_MESSAGE =

```
 ___
| TRANSIENT_ERROR_THRESHOLD_COUNTER_OVERFLOW |
| FLOW_CONTROL_ERRORS                        |
| OUTGOING_DIALOGUE_DATA_PACKET              |
| RECEIVED_INCORRECT_PASSWORD_COMMAND        |
| ERROR_REASON                               |
 --                                          --
```

NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA FLOW NAME: OUTGOING_NODE_NETWORK_PACKET
ALIASES: SECONDARY_OUTGOING_NODE_NETWORK_PACKET
COMPOSITION:

OUTGOING_NODE_NETWORK_PACKET =

```
 ___
| ADJACENT_NODE_ROUTING_PACKET |
| TRANSPORT_TO_NETWORK_PACKET  |
| OLD_ROUTING_PACKETS          |
| HOPPED_ROUTING_PACKET        |
| VALID_NETWORK_PACKET         |
| INITIAL_ROUTE_PACKET         |
 --                            --
```

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA FLOW NAME: OUTGOING_NODE_TRANSPORT_PACKET
ALIASES: NONADJACENT_ROUTE_PACKET
OUTGOING_TRANSPORT_ROUTE_PACKET
SECONDARY_OUTGOING_NODE_TRANSPORT_PACKET
TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA FLOW NAME: OUTGOING_TRANSPORT_ROUTE_PACKET
ALIASES: NONADJACENT_ROUTE_PACKET
OUTGOING_NODE_TRANSPORT_PACKET
SECONDARY_OUTGOING_NODE_TRANSPORT_PACKET
TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: OUTGOING_SATELLITE_TRANSPORT_PACKET
ALIASES: SECONDARY_OUTGOING_SATELLITE_TRANSPORT_PACKET
COMPOSITION:

OUTGOING_SATELLITE_TRANSPORT_PACKET =

```
                    ---                          ---
                   | NONADJACENT_SATELLITE_PACKET |
                   | ADJACENT_NODE_PACKETS        |
                    ---                          ---
```

NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
OVERVIEW LAYER

DATA ELEMENT NAME: PASSWORD
ALIASES: NONE
VALUES AND MEANINGS:
THE PASSWORD FOR THE REQUESTING NODE. IT IS THE SAME FOR THE ENTIRE NETWORK.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: PASSWRD
ALIASES: NONE
VALUES AND MEANINGS:
ACCESS VERIFICATION PASSWORD = AN ARBITRARY BYTE STRING USED FOR CROSS-CHECK VERIFICATION (NORMALLY UNIQUELY PAIRED WITH THE REQUESTOR-ID OR WITH THE SERVICE).
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: PERSISTENT_ERROR
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE CONNECT PACKET LAYER

DATA FLOW NAME: PIGGYBACKED_TRANSPORT_DATA_SEGMENT
ALIASES: INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT

TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET
ALIASES: INCOMING_DIALOGUE_PACKET
TRANSPORT_INTERRUPT_PACKET
VALID_INTERRUPT_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE I/L PACKET LAYER

DATA ELEMENT NAME: REASON
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO FILE BY SAME NAME.

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: RECEIVED_ADJACENT_NODE_PACKETS
ALIASES: VALID_SATELLITE_TRANSPORT_PACKET
VALID_DSTADDR_PACKETS
VALID_DSTNODE_PACKETS
VALID_DSTNODE_CI_PACKET

COMPOSITION:
RECEIVED_ADJACENT_NODE_PACKETS =

```
 ___                                                  ___
|   RECEIVED_TRANSPORT_INITIALIZATION_PACKET          |
|   RECEIVED_TRANSPORT_DATA_PACKET                    |
|   RECEIVED_TRANSPORT_CONTROL_PACKET                 |
|   RECEIVED_TRANSPORT_ACKNOWLEDGE_PACKET             |
 --                                                  --
```

NOTES: DECODE ROUTE HEADER LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_CONNECT_CONFIRM_PACKET
ALIASES: CONNECT_CONFIRM_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE CONNECT PACKET LAYER

DATA FLOW NAME: RECEIVED_CONNECT_INITIATE_PACKET
ALIASES: CONNECT_INITIATE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: RECEIVED_CORRECT_PASSWORD_COMMAND
ALIASES: CORRECT_PASSWORD_COMMAND
VALUES AND MEANINGS:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: RECEIVED_DISCONNECT_CONFIRM_PACKET
ALIASES: FIRST_DISCONNECT_CONFIRM_PACKET
DISCONNECT_CONFIRM_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER

DATA FLOW NAME: RECEIVED_DISCONNECT_INITIATE_PACKET
ALIASES: DISCONNECT_INITIATE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER
EXECUTE CONTROL PACKET LAYER

DATA FLOW NAME: RECEIVED_I/L_ACK_PACKET
ALIASES: NONE
COMPOSITION:
RECEIVED_I/L_ACK_PACKET = | I/L_ACK_PACKET |
| I/L_NAK_PACKET |
NOTES: EXECUTE ACKNOWLEDGE PACKET LAYER

DATA FLOW NAME: RECEIVED_I/L_PACKET

ALIASES: VALID_I/L_PACKET
COMPOSITION:

RECEIVED_I/L_PACKET = [VALID_LINK_SERVICES_PACKET | VALID_INTERRUPT_PACKET]

NOTES: EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: RECEIVED_INCORRECT_PASSWORD_COMMAND
ALIASES: INCORRECT_PASSWORD_COMMAND
VALUES AND MEANINGS:

SEE ALIASES

NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_INTERCEPT_INITIALIZATION_PACKET
ALIASES: RECEIVED_NODE_INITIALIZATION_PACKET
RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET
NODE_INITIALIZATION_V0_PACKET
NODE_INITIALIZATION_V1_PACKET
COMPOSITION:

SEE ALIASES

NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: RECEIVED_NODE_INITIALIZATION_PACKET
ALIASES: RECEIVED_INTERCEPT_INITIALIZATION_PACKET
RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET
NODE_INITIALIZATION_V0_PACKET
NODE_INITIALIZATION_V1_PACKET
COMPOSITION:

RECEIVED_NODE_INITIALIZATION_PACKET = MSGFLG + STARTTYPE + NODEADDR + NODENAME + FUNCTIONS + REQUESTS + BLKSIZE + NSPSIZE + MAXLINKS + ROUTVER + COMMVER + SYSVER

NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: RECEIVED_NODE_VERIFICATION_PACKET
ALIASES: NODE_VERIFICATION_PACKET
COMPOSITION:

RECEIVED_NODE_VERIFICATION_PACKET = MSGFLG + STARTTYPE + PASSWORD

NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET
ALIASES: RECEIVED_NODE_INITIALIZATION_PACKET
NODE_INITIALIZATION_V0_PACKET
NODE_INITIALIZATION_V1_PACKET
RECEIVED_INTERCEPT_INITIALIZATION_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE STARTUP PACKET LAYER

DATA FLOW NAME: RECEIVED_NORMAL_DATA_ACK_PACKET
ALIASES: NONE
COMPOSITION:

RECEIVED_NORMAL_DATA_ACK_PACKET = | DATA_ACK_PACKET |
| DATA_NAK_PACKET |

NOTES: EXECUTE ACKNOWLEDGE PACKET LAYER

DATA FLOW NAME: RECEIVED_NORMAL_DATA_SEGMENT
ALIASES: INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: RECEIVED_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: RECEIVED_TRANSPORT_ACKNOWLEDGE_PACKET
ALIASES: TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE ACKNOWLEDGE PACKET LAYER

DATA FLOW NAME: RECEIVED_TRANSPORT_CONTROL_PACKET
ALIASES: TRANSMITTED_TRANSPORT_CONTROL_PACKET
COMPOSITION:

SEE ALIASES
NOTES: EXECUTE CONNECT PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_TRANSPORT_DATA_PACKET
ALIASES: NONE
COMPOSITION:

RECEIVED_TRANSPORT_DATA_PACKET =

| RECEIVED_NORMAL_DATA_SEGMENT |
| RECEIVED_I/L_PACKET |

NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER

DATA FLOW NAME: RECEIVED_TRANSPORT_INITIALIZATION_PACKET
ALIASES: NONE
COMPOSITION:

RECEIVED_TRANSPORT_INITIALIZATION_PACKET =

| RECEIVED_INCORRECT_PASSWORD_COMMAND |
| RECEIVED_NODE_INITIALIZATION_PACKET |
| RECEIVED_NODE_VERIFICATION_PACKET |
| RECEIVED_CORRECT_PASSWORD_COMMAND |

NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: REJECT_CONFIRM
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THE REJECTION OF A RECEIVED_
CONNECT_CONFIRM_PACKET. RESULTS IN A
DISCONNECT_INITIATE_PACKET BEING GENERATED.
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: REJECT_CONNECT
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO INDICATE THE REJECTION OF A RECEIVED_ CONNECT_INITIATE_PACKET. RESULTS IN THE GENERATION OF A DISCONNECT_INITIATE_PACKET.

NOTES: EXECUTE CONNECT PACKET LAYER

DATA FLOW NAME: RETRANSMITTED_DATA_PACKET
ALIASES: INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET

COMPOSITION:

SEE ALIASES

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: RETRANSMITTED_I/L_PACKET
ALIASES: COUNTED_TRANSPORT_I/L_PACKET
TRANSPORT_I/L_PACKET
TRANSMITTED_I/L_PACKET

COMPOSITION:

SEE ALIASES

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA ELEMENT NAME: RETURNING
ALIASES: NONE
VALUES AND MEANINGS:

THIS BIT IS TURNED ON WHEN A PACKET IS ON ITS RETURN JOURNEY. NOT PRESENTLY USED.

NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: RETURN_REQUEST
ALIASES: NONE
VALUES AND MEANINGS:

THIS BIT INDICATES THAT AN UNDELIVERABLE PACKET SHOULD BE RETURNED TO THE SENDER.

NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: REQUESTS
ALIASES: NONE
VALUES AND MEANINGS:

THE REQUESTS DESIRED OF THE RECEIVER BY THE SENDER OF THE INITIALIZATION PACKET. FORMAT:

VERIF = NODE VERIFICATION PACKET REQUIRED.
RINT = 0 = NO INTERCEPT = SET ON TRANSMIT BY A ROUTING NODE GOING TO A SATELLITE NODE.
= 1 = INTERCEPT = SET ON TRANSMIT BY A ROUTING NODE GOING TO A ROUTING NODE OR ON TRANSMIT BY A SATELLITE NODE GOING TO A ROUTING NODE.

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: ROUTING_PACKET
ALIASES: VALID_ROUTING_PACKET
HOPPED_ROUTING_PACKET
INITIAL_ROUTE_PACKET
ADAJACENT_NODE_ROUTING_PACKET
OLD_ROUTING_PACKETS
COMPOSITION:
ROUTING_PACKET = DESTINATION + LINE_COST + HOP_COUNT
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: ROUTVER
ALIASES: NONE
VALUES AND MEANINGS:

THE VERSION OF THE ROUTING ALGORITHM. FORMAT:

1. VERSION NUMBER
2. ECO NUMBER
3. CUSTOMER LEVEL NUMBER

NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: RTHDR

ALIASES: NONE
VALUES AND MEANINGS:

THE RTHDR IS USED BY NODES CONTAINING AN INTERCEPT

FUNCTION

TO DETERMINE THE PHYSICAL LINK ON WHICH TO SEND THE PACKET TOWARD THE DESTINATION NODE. A TRANSPORT PACKET GOING TO AN ADJACENT NODE WILL NOT CONTAIN THE RTHDR FIELD UNLESS IT IS A NETWORK_TO_NETWORK_PACKET. FORMAT:

1. DSTNODE = DESTINATION NODE NAME
2. SRCNODE = SOURCE NODE NAME
3. MPRI = PRIORITY

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: RTHDR_NONADJACENT_NODE_PACKETS
ALIASES: NONE
COMPOSITION:

RTHDR_NONADJACENT_NODE_PACKETS = NONADJACENT_NODE_PACKETS + RTHDR FIELD

NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA ELEMENT NAME: RQSTRID
ALIASES: NONE
VALUES AND MEANINGS:

REQUESTOR ID = A CHARACTER-CODED REFERENCE THAT, IN A SINGLE NODE CONTEXT, UNIQUELY IDENTIFIES THE PERSON OR PROCESS REQUESTING SERVICE.

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: SECONDARY_INCOMING_NODE_NETWORK_PACKET
ALIASES: INCOMING_NODE_NETWORK_PACKET
COMPOSITION:

SEE ALIASES

NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_INCOMING_NODE_TRANSPORT_PACKET
ALIASES: INCOMING_SATELLITE_TRANSPORT_PACKET
SECONDARY_INCOMING_SATELLITE_TRANSPORT_PACKET
INCOMING_NODE_TRANSPORT_PACKET
COMPOSITION:

SEE ALIASES

NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_INCOMING_SATELLITE_TRANSPORT_PACKET
ALIASES: INCOMING_SATELLITE_TRANSPORT_PACKET
INCOMING_NODE_TRANSPORT_PACKET
SECONDARY_INCOMING_NODE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_DIALOGUE_MESSAGE
ALIASES: OUTGOING_DIALOGUE_MESSAGE
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_NODE_NETWORK_PACKET
ALIASES: OUTGOING_NODE_NETWORK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_NODE_TRANSPORT_PACKET
ALIASES: *NONADJACENT_ROUTE_PACKET*
OUTGOING_TRANSPORT_ROUTE_PACKET
OUTGOING_NODE_TRANSPORT_PACKET
TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_SATELLITE_TRANSPORT_PACKET
ALIASES: OUTGOING_SATELLITE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: SEGMENT_COUNT
ALIASES: NONE
VALUES AND MEANINGS:
THE NUMBER OF SEGMENTS THE TRANSMITTER IS GOING TO BREAK THE INCOMING DIALOGUE DATA INTO.

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA ELEMENT NAME: SEGNUM
ALIASES: NONE
VALUES AND MEANINGS:
THE NUMBER OF THIS SEGMENT, MODULO 4096.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: SEGNUMI
ALIASES: NONE
VALUES AND MEANINGS:
THE NUMBER OF THIS INTERRUPT OR LINK SERVICE PACKET. NUMBERS FOR INTERRUPT OR LINK SERVICE PACKETS WILL HAVE NO RELATIONSHIP TO THE NUMBERS ASSIGNED TO NORMAL DATA PACKETS. EACH PACKET TYPE UTILIZES A DIFFERENT SUBCHANNEL ON A LOGICAL LINK.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: SEGSIZE
ALIASES: NONE
VALUES AND MEANINGS:
THE MAXIMUM SIZE OF A NORMAL DATA SEGMENT TO BE RECEIVED ON THIS LOGICAL LINK.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: SERVICES
ALIASES: NONE
VALUES AND MEANINGS:
REQUESTED SERVICES. FORMAT:
FCOPT = FLOW CONTROL OPTION
= 0 = NONE
= 1 = SEGMENT REQUEST COUNT
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: SOURCE
ALIASES: NONE
VALUES AND MEANINGS:
THE NAME OF THE NODE FROM WHICH THE ROUTING PACKET CAME FROM.

NOTES: ALL NETWORK LAYERS

DATA ELEMENT NAME: SRCADDR
ALIASES: NONE
VALUES AND MEANINGS:
THE LOGICAL LINK SOURCE ADDRESS. THIS ADDRESS IS ASSIGNED WHEN A LINK IS ESTABLISHED (CONNECT_PACKETS).
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: SRCNAME
ALIASES: NONE
VALUES AND MEANINGS:
THE SOURCE PROCESS IDENTIFICATION. USE OF FORMAT 02 AS DESCRIBED IN DSTNAME.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: START
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT THE NODE SHOULD START THE INITIALIZATION PROCESS.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: START_DATA
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO TRIGGER THE RETRANSMITTING OF NEGATIVELY ACKNOWLEDGED DATA PACKETS.
NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA ELEMENT NAME: START_I/L
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO TRIGGER THE RETRANSMITTING OF NEGATIVELY ACKNOWLEDGED INTERRUPT AND LINK SERVICES PACKETS.
NOTES: EXECUTE I/L PACKET LAYER

DATA ELEMENT NAME: STARTTYPE
ALIASES: NONE

VALUES AND MEANINGS:
TYPE OF STARTUP MESSAGE. 1 = NODE_INITIALIZATION_PACKET
2 = NODE_VERIFICATION_PACKET
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: STOP
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO START LOGICAL LINK DISCONNECTION DUE TO A RECEIVED_DISCONNECT_REQUEST_COMMAND.
NOTES: EXECUTE CONNECT PACKET LAYER

DATA ELEMENT NAME: STOP_LINK
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE DISCONNECT PACKET LAYER

DATA ELEMENT NAME: STOP_LINK_NOW
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT A DISCONNECT_CONFIRM_PACKET CAN NOW BE ISSUED SINCE ALL DATA_PACKETS SENT OUT HAVE BEEN POSITIVELY ACKNOWLEDGED.
NOTES: EXECUTE DISCONNECT PACKET LAYER

DATA ELEMENT NAME: SYSVER
ALIASES: NONE
VALUES AND MEANINGS:
A STRING DESCRIBING THE OPERATING SYSTEM, DATA OF CREATION, ETC.
NOTES: ALL TRANSPORT LAYERS

DATA ELEMENT NAME: TRANSIENT_ERROR_THRESHOLD_COUNTER_OVERFLOW
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: TRANSMIT_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.
NOTES: EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: TRANSMITTED_DATA_PACKET
ALIASES: INCOMING_DIALOGUE_PACKET
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
RETRANSMITTED_DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: TRANSMITTED_I/L_PACKET
ALIASES: COUNTED_TRANSPORT_I/L_PACKET
TRANSPORT_I/L_PACKET
RETRANSMITTED_I/L_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA FLOW NAME: TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET
ALIASES: RECEIVED_TRANSPORT_ACKNOWLEDGE_PACKET
COMPOSITION:
TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET =

DATA_ACK_PACKET
DATA_NAK_PACKET
I/L_ACK_PACKET
I/L_NAK_PACKET

NOTES: EXECUTE DATA PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: TRANSMITTED_TRANSPORT_DATA_PACKET
ALIASES: NONE
COMPOSITION:

TRANSMITTED_TRANSPORT_DATA_PACKET =

```
                    | RETRANSMITTED_DATA_PACKET |
                    | RETRANSMITTED_I/L_PACKET  |
                    | TRANSMITTED_I/L_PACKET    |
                    | TRANSMITTED_DATA_PACKET   |
```

NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

DATA FLOW NAME: TRANSMITTED_TRANSPORT_CONTROL_PACKET
ALIASES: RECIEVED_TRANSPORT_CONTROL_PACKET
COMPOSITION:

TRANSMITTED_TRANSPORT_CONTROL_PACKET =

```
                    | DISCONNECT_INITIATE_PACKET |
                    | CONNECT_INITIATE_PACKET    |
                    | CONNECT_CONFIRM_PACKET     |
                    | DISCONNECT_CONFIRM_PACKET  |
```

NOTES: EXECUTE CONNECT PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: TRANSMITTED_TRANSPORT_INITIALIZATION_PACKET
ALIASES: NONE
COMPOSITION:

TRANSMITTED_TRANSPORT_INITIALIZATION_PACKET =

```
                  | NODE_INITIALIZATION_V0_PACKET |
                  | NODE_INITIALIZATION_V1_PACKET |
                  | NODE_VERIFICATION_PACKET      |
                  | CORRECT_PASSWORD_COMMAND      |
                  | INCORRECT_PASSWORD_COMMAND    |
```

NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER
EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: TRANSPORT_DATA_SEGMENT
ALIASES: INCOMING_DIALOGUE_DATA
NORMAL_DATA_PACKET
VALID_NORMAL_DATA_SEGMENT
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
RETRANSMITTED_DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DIALOGUE SEGMENT LAYER

DATA FLOW NAME: TRANSPORT_INTERRUPT_PACKET
ALIASES: INCOMING_DIALOGUE_INTERRUPT
PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET
VALID_INTERRUPT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE I/L PACKET LAYER

DATA FLOW NAME: TRANSPORT_PACKET
ALIASES: NONADJACENT_ROUTE_PACKET
OUTGOING_TRANSPORT_ROUTE_PACKET
OUTGOING_NODE_TRANSPORT_PACKET
SECONDARY_OUTGOING_NODE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: TRANSPORT_TO_NETWORK_PACKET
ALIASES: NONE
COMPOSITION:
TRANSPORT_TO_NETWORK_PACKET = TRANSPORT_PACKET + HOPS +
SOURCE + DESTINATION +
CHOKE + RETURN + REQUEST +
RETURNING + VERSION
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: TSTDATA
ALIASES: NONE
VALUES AND MEANINGS:

ANY TEST DATA PATTERN. THIS PACKET IS IGNORED ON RECEIVE.

NOTES: ALL TRANSPORT LAYERS

DATA FLOW NAME: VALID_DSTADDR_PACKETS
ALIASES: RECEIVED_ADJACENT_NODE_PACKETS
VALID_SATELLITE_TRANSPORT_PACKETS
VALID_DSTNODE_PACKETS
VALID_DSTNODE_CI_PACKET
COMPOSITION:
SEE ALIASES
NOTES: DECODE ROUTE HEADER LAYER

DATA FLOW NAME: VALID_DSTNODE_CI_PACKET
ALIASES: RECEIVED_ADJACENT_NODE_PACKETS
VALID_SATELLITE_TRANSPORT_PACKET
VALID_DSTADDR_PACKETS
VALID_DSTNODE_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER
DECODE ROUTE HEADER LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER

DATA FLOW NAME: VALID_DSTNODE_PACKETS
ALIASES: RECEIVED_ADJACENT_NODE_PACKETS
VALID_SATELLITE_TRANSPORT_PACKET
VALID_DSTADDR_PACKETS
VALID_DSTNODE_CI_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: DECODE ROUTE HEADER LAYER

DATA FLOW NAME: VALID_I/L_PACKET
ALIASES: RECEIVED_I/L_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_INTERRUPT_PACKET
ALIASES: INCOMING_DIALOGUE_INTERRUPT
TRANSPORT_INTERRUPT_PACKET

PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_LINK_SERVICES_PACKET
ALIASES: INTERRUPT_LINK_SERVICES_PACKET
DATA_LINK_SERVICES_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_NETWORK_PACKET
ALIASES: NETWORK_TO_NETWORK_PACKET
VALID_NETWORK_TO_NETWORK_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: VALID_NETWORK_TO_NETWORK_PACKET
ALIASES: NETWORK_TO_NETWORK_PACKET
VALID_NETWORK_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: VALID_NORMAL_DATA_SEGMENT
ALIASES: INCOMING_DIALOGUE_PACKET
NORMAL_DATA_PACKET
RECEIVED_NORMAL_DATA_SEGMENT
INCOMING_DIALOGUE_SEGMENT
TRANSPORT_DATA_SEGMENT
PIGGYBACKED_TRANSPORT_DATA_SEGMENT
COUNTED_TRANSPORT_DATA_SEGMENT
TRANSMITTED_DATA_PACKET
RETRANSMITTED_DATA_PACKET

COMPOSITION:
SEE ALIASES

NOTES: EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_ROUTING_PACKET
ALIASES: ROUTING_PACKET
HOPPED_ROUTING_PACKET
INITIAL_ROUTE_PACKET
ADJACENT_NODE_ROUTING_PACKET
OLD_ROUTING_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: VALID_SATELLITE_TRANSPORT_PACKET
ALIASES: RECEIVED_ADJACENT_NODE_PACKETS
VALID_DSTADDR_PACKETS
VALID_DSTNODE_PACKETS
VALID_DSTNODE_CI_PACKET
COMPOSITION:
SEE ALIASES
NOTES: DECODE ROUTE HEADER LAYER

DATA ELEMENT NAME: VERIFY_CLEAR
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG TO INDICATE THAT A
RECEIVED_INTERCEPT_INITIALIZATION_PACKET
WAS RECEIVED WITH THE VERIFY BIT NOT SET CAUSING
GENERATION OF ANOTHER RECEIVED_INTERCEPT_INITIALIZATION_
PACKET WITH THE VERIFY BIT NOT SET.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: VERIFY_SET
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO INDICATE THAT A RECEIVED_INTERCEPT_
INITIALIZATION_PACKET WAS RECEIVED WITH THE VERIFY BIT
SET CAUSING THE GENERATION OF A NODE_VERIFICATION_PACKET.
NOTES: EXECUTE STARTUP PACKET LAYER

DATA ELEMENT NAME: VERSION
ALIASES: NONE
VALUES AND MEANINGS:
THIS BIT TELLS WHICH VERSION OF THE NETWORK PROTOCOL
GENERATED THE PACKET. PRESENTLY NOT USED.
NOTES: ALL NETWORK LAYERS

FILE DEFINITIONS (T/N)

FILE OR DATABASE NAME: ACK_D_FILE
ALIASES: NONE
COMPOSITION:

ACK_D_FILE = THE SEGMENT NUMBER OF THE LAST DATA PACKET POSITIVELY ACKNOWLEDGED.

ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE DATA PACKET LAYER
EXECUTE ACKNOWLEDGE PACKET LAYER

FILE OR DATABASE NAME: ACK_I/L_FILE
ALIASES: NONE
COMPOSITION:

ACK_I/L_FILE = THE SEGMENT NUMBER OF THE LAST INTERRUPT OR LINK SERVICES PACKET POSITIVELY ACKNOWLEDGED TO THIS NODE.

ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE I/L PACKET LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER
EXECUTE DATA PACKET LAYER
EXECUTE ACKNOWLEDGE PACKET LAYER

FILE OR DATABASE NAME: ADJACENT_NODE_PARAMETERS
ALIASES: NONE
COMPOSITION:

ADJACENT_NODE_PARAMETERS = NODEADDR + NODENAME + NO INTERCEPT + INTERCEPT REQUESTED + BLKSIZE + NSPSIZE + MAXLNKS + COMMVER + SYSVER + CORRECT_ PASSWORD + LINK_ACCESSABLE

ORGANIZATION: VARIABLE LENGTH 8-BIT BYTE COMBINATIONS
NOTES: EXECUTE STARTUP PACKET LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE INCOMING DIALOGUE MESSAGE LAYER

FILE OR DATABASE NAME: DATA_MEMORY
ALIASES: NONE
COMPOSITION:

DATA_MEMORY = DATA SEGMENTS ARE LOADED INTO MEMORY UNTIL SUCH TIME A POSITIVE ACKNOWLEDGEMENT IS RECEIVED.

ORGANIZATION: ARRAY OF VARIABLE BYTE LENGTH WORDS.
NOTES: EXECUTE DIALOGUE SEGMENT LAYER

FILE OR DATABASE NAME: DATA_NAK_FILE
ALIASES: NONE
COMPOSITION:

DATA_NAK_FILE = THE SEGMENT NUMBER OF THE LAST DATA PACKET NEGATIVELY ACKNOWLEDGED TO THIS NODE, AT OTHER TIMES IT IS EQUAL TO TRANSMIT_D_FILE + 1.

ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE DATA PACKET LAYER
EXECUTE ACKNOWLEDGE PACKET LAYER

FILE OR DATABASE NAME: DIALOGUE_PROCESS_TABLE
ALIASES: NONE
COMPOSITION:

DIALOGUE_PROCESS_TABLE = DESTINATION NODE NAME + DESTINATION PROCESS IDENTIFICATION + SOURCE PROCESS IDENTIFICATION + LINK IDENTIFIER + ACCESS CONTROL INFORMATION + SOURCE NODE NAME + SOURCE PROCESS IDENTIFICATION + DESTINATION PROCESS IDENTIFICATION + REPLY IDENTIFIER + REMOTE PROCESS'S SEGMENT SIZE.

ORGANIZATION: VARIABLE LENGTH 8-BIT BYTE WORDS.
NOTES: EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER
EXECUTE DIALOGUE SEGMENT LAYER
DECODE ROUTE HEADER LAYER
EXECUTE DATA PACKET LAYER
EXECUTE I/L PACKET LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER

```
FILE OR DATABASE NAME:  FLOW_CONTROL_PARAMETERS
ALIASES:                NONE
COMPOSITION:
                        FLOW_CONTROL_PARAMETERS = DATA FLOW CONTROL SWITCH +
                                                  INTERRUPT REQUEST COUNT +
                                                  DATA REQUEST SWITCH + DATA
                                                  REQUEST COUNT
ORGANIZATION:           INITIAL PARAMETERS = OPEN,1,SEGMENT,0   RESPECTIVELY
NOTES:                  EXECUTE DIALOGUE SEGMENT LAYER
                        EXECUTE INCOMING DIALOGUE MESSAGE LAYER
                        EXECUTE I/L PACKET LAYER
                        EXECUTE DATA PACKET LAYER
                        EXECUTE CONTROL PACKET LAYER
                        EXECUTE CONNECT PACKET LAYER


FILE OR DATABASE NAME:  I/L_MEMORY
ALIASES:                NONE
COMPOSITION:
                        I/L_MEMORY = INTERRUPT AND LINK SERVICE PACKETS ARE
                                     LOADED INTO MEMORY UNTIL SUCH TIME A
                                     POSITIVE ACKNOWLEDGEMENT IS RECEIVED.
ORGANIZATION:           ARRAY OF VARIABLE BYTE LENGTH WORDS.
NOTES:                  EXECUTE I/L PACKET LAYER


FILE OR DATABASE NAME:  I/L_NAK_FILE
ALIASES:                NONE
COMPOSITION:
                        I/L_NAK_FILE = THE SEGMENT NUMBER OF THE LAST
                        INTERRUPT OR LINK SERVICES PACKET NEGATIVELY
                        ACKNOWLEDGED TO
                        THIS NODE. ALSO EQUAL TO TRANSMIT_I/L_FILE + 1.
ORGANIZATION:           1 8-BIT BYTE
NOTES:                  EXECUTE I/L PACKET LAYER
                        EXECUTE DATA PACKET LAYER
                        EXECUTE ACKNOWLEDGE PACKET LAYER


FILE OR DATABASE NAME:  INITIAL_PARAMETER_LIST
ALIASES:                NONE
COMPOSITION:
                        INITIAL_PARAMETER_LIST = NODEADDR + NODENAME +

                                ___                ___
                               | NO_INTERCEPT | + | NO_INTERCEPT_REQUESTED |
                               | INTERCEPT    |   | INTERCEPT_REQUESTED    |
```

```
                                    --          --   --                    --
                                    + BLKSIZE + NSPSIZE + MAXLNKS + ROUTVER +
                                    COMMVER + SYSVER
ORGANIZATION:           VARIABLE LENGTH 8-BIT BYTE COMBINATIONS
NOTES:                  EXECUTE STARTUP PACKET LAYER


FILE OR DATABASE NAME:  LINK_TO_TRANSPORT_COMMAND_TABLE
ALIASES:                NONE
COMPOSITION:
                        REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL
ORGANIZATION:           SINGLE VARIABLE
NOTES:                  EXECUTE STARTUP PACKET LAYER
                        EXECUTE CONNECT PACKET LAYER
                        EXECUTE TRANSPORT PROTOCOL AT PRIMARY NODE LAYER


FILE OR DATABASE NAME:  LS_FILE
ALIASES:                NONE
COMPOSITION:
                        LS_FILE = SEGMENT NUMBER OF LAST PACKET SENT.
ORGANIZATION:           1 BYTE
NOTES:                  EXECUTE DIALOGUE SEGMENT LAYER
                        EXECUTE INCOMING DIALOGUE MESSAGE LAYER
                        EXECUTE CONTROL PACKET LAYER
                        EXECUTE CONNECT PACKET LAYER
                        EXECUTE DISCONNECT PACKET LAYER


FILE OR DATABASE NAME:  REASON
ALIASES:                NONE
COMPOSITION:
                        A FILE WITH THE FOLLOWING ERRORS:
                    ERROR CODE      MEANING
                    ----------      -------
                         0      NO ERROR
                         1      RESOURCES ALLOCATION FAILURE
                         2      DESTINATION NODE DOES NOT EXIST
                         3      NODE SHUTTING DOWN
                         4      DESTINATION PROCESS DOES NOT EXIST
                         5      INVALID PROCESS NAME FIELD
                         6      DESTINATION PROCESS QUEUE OVERFLOW
                         7      UNSPECIFIED ERROR CONDITION
                         8      THIRD PARTY ABORTED THE LOGICAL LINK
                         9      LINK ABORT BY DIALOGUE PROCESS
                        10      FLOW CONTROL VIOLATION-ILLEGAL FCVAL IN LINK
                                SERVICES MESSAGE
                        11      TOO MANY CONNECTIONS TO NODE
```

12 TOO MANY CONNECTIONS TO DESTINATION PROCESS
13 ACCESS NOT PERMITTED-UNACCEPTABLE RQSTRID OR PASSWORD
14 LOGICAL LINK SERVICES MISMATCH
15 UNACCEPTABLE ACCOUNT INFORMATION-UNAUTHORIZED OR ACCOUNT BALANCE UNACCEPTABLE
16 SEGSIZE TOO SMALL
17 DIALOGUE PROCESS ABORTED, TIMED OUT, OR CANCELLED REQUEST
18 NO PATH TO DESTINATION NODE
19 FLOW CONTROL FAILURE
20 DSTADDR LOGICAL LINK DOES NOT EXIST
21 CONFIRMATION OF DISCONNECT INITIATE
22 IMAGE DATA FIELD TOO LONG-RQSTRID, PASSWORD, ACCOUNT, USRDATA, AND DATA.

ORGANIZATION: EACH ERROR = 2 BYTES
NOTES: DECODE ROUTE HEADER LAYER
EXECUTE CONTROL PACKET LAYER
EXECUTE CONNECT PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER

FILE OR DATABASE NAME: RECEIVED_D_FILE
ALIASES: NONE
COMPOSITION:
RECEIVED_D_FILE = MOST CURRENT DATA SEGMENT NUMBER RECEIVED.
ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE DATA PACKET LAYER

FILE OR DATABASE NAME: RECEIVED_I/L_FILE
ALIASES: NONE
COMPOSITION:
RECEIVED_I/L_FILE = MOST CURRENT I/L SEGMENT NUMBER RECEIVED.
ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE DATA PACKET LAYER

FILE OR DATABASE NAME: ROUTING_TABLE_1
ALIASES: NONE
COMPOSITION:
ROUTING_TABLE_1 = 1-2 = 6
2-3 = 5
3-1 = 10
1-2-3 = 11
2-3-1 = 15

3-1-2 = 16
ABOVE NUMBERS HAVE ASSOCIATED NODE NAMES.

ORGANIZATION: THIS IS A TABLE THAT CONTAINS THE PERMINATE LINK COST, IF THE FULL NETWORK WERE UP AND RUNNING. THE ABOVE VALUES ARE FOR A 3 ROUTING NODE NETWORK CONFIGURATION.

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

FILE OR DATABASE NAME: ROUTING_TABLE_2
ALIASES: NONE
COMPOSITION:

ROUTING_TABLE_2 = 1-2 + 2-3 + 3-1 + 1-2-3 + 2-3-1 + 3-1-2

ORGANIZATION: THIS IS THE TABLE THAT IS ACTUALLY USED AS THE NETWORK ROUTING TABLE FOR GENERATING AND SENDING ROUTING PACKETS.

NOTES: EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

FILE OR DATABASE NAME: SATELLITE_NODE_PARAMETERS
ALIASES: NONE
COMPOSITION:

SATELLITE_NODE_PARAMETERS = NODEADDR + NODENAME + INTERCEPT + NO INTERCEPT + BLKSIZE + NSPSIZE + MAXLNKS + ROUTVER + COMMVER + SYSVER + PASSWORD

ORGANIZATION: VARIABLE LENGHT 8-BIT BYTE COMBINATIONS

NOTES: EXECUTE STARTUP PACKET LAYER
DECODE ROUTE HEADER LAYER

FILE OR DATABASE NAME: TRANSMIT_D_FILE
ALIASES: NONE
COMPOSITION:

TRANSMIT_D_FILE = DATA TRANSMITTING SEGMENT NUMBER, MODULO 4096.

ORGANZATION: 1 8-BIT BYTE

NOTES: EXECUTE DIALOGUE SEGMENT LAYER

FILE OR DATABASE NAME: TRANSMIT_I/L_FILE
ALIASES: NONE
COMPOSITION:

TRANSMIT_I/L_FILE = INTERRUPT OR LINK SERVICES TRANSMITTING SEGMENT NUMBER,

MODULO 4096.

ORGANIZATION: 1 8-BIT BYTE
NOTES: EXECUTE I/L PACKET LAYER

FILE OR DATABASE NAME: TRANSPORT_TO_LINK_COMMAND_TABLE
ALIASES: NONE
COMPOSITION:

REFER TO DATA DICTIONARY FOR LINK LEVEL PROTOCOL.

ORGANIZATION: SINGLE VARIABLE
NOTES: EXECUTE DIALOGUE SEGMENT LAYER
EXECUTE STARTUP PACKET LAYER
EXECUTE DISCONNECT PACKET LAYER
EXECUTE OUTGOING TRANSPORT PACKETS LAYER
EXECUTE NETWORK PROTOCOL AT PRIMARY NODE LAYER

PROCESS SPECIFICATIONS (T/N)

```
PROCESS NAME:          DECODE RTHDR FIELD
PROCESS NUMBER:        1.1.1
PROCESS DESCRIPTION:
WHEN Input = INCOMING_SATELLITE_TRANSPORT_PACKET or INCOMING_NODE_TRANSPORT_
     PACKET then
          Check for RTHDR field
               IF RTHDR field not present then
                    Output RECEIVED_ADJACENT_NODE_PACKETS
               ELSE
                    IF DSTNODE = valid satellite node and MSGFLG = 00011000
                              then
                         Output VALID_DST_CI_PACKET
                    ELSEIF DSTNODE = valid satellite node then
                         Output VALID_DSTNODE_PACKETS
                    ELSEIF DSTNODE = valid routing node then
                         Output OUTGOING_TRANSPORT_ROUTE_PACKET
                    ELSE output INVALID_DSTNODE_PACKETS


PROCESS NAME:          DECODE MSGFLG FIELD
PROCESS NUMBER:        1.1.2
PROCESS DESCRIPTION:
IF MSGFLG field = valid destination logical link address then
     Output VALID_DSTADDR_PACKETS
ELSE output INVALID_DSTADDR_PACKETS


PROCESS NAME:          GENERATE FIRST DISCONNECT CONFIRM
PROCESS NUMBER:        1.1.3
PROCESS DESCRIPTION:
IF Input = INVALID_DSTADDR_PACKETS or DISCONNECT_REQUIRED or INVALID_DSTNODE_
           PACKETS then
     Extract REASON_CODE from REASON file and
     Output FIRST_DISCONNECT_CONFIRM_PACKET
ELSE Null


PROCESS NAME:          EXAMINE LOGICAL LINK DATABASE
PROCESS NUMBER:        1.1.4
PROCESS DESCRIPTION:
IF SATELLITE_NODE_PARAMETERS are null then
     Output DISCONNECT_REQUIRED
ELSE output VALID_SATELLITE_TRANSPORT_PACKET
```

```
PROCESS NAME:          PASS TO SATELLITE
PROCESS NUMBER:        1.1.5
PROCESS DESCRIPTION:
WHEN Input is VALID_SATELLITE_TRANSPORT_PACKET or VALID_DSTNODE_CI_PACKET
     Pass to satellite node according to DSTADDR value.
ELSE Null


PROCESS NAME:          DECODE PACKET TYPE
PROCESS NUMBER:        1.2
PROCESS DESCRIPTION:
WHEN Input is RECEIVED_ADJACENT_NODE_PACKETS then
     IF MSGFLG = 0BB00000 where BB = 00,01,10,11 then
          Output RECEIVED_TRANSPORT_DATA_PACKET
     ELSEIF MSGFLG = 00110000 then
          Output RECEIVED_TRANSPORT_DATA_PACKET
     ELSEIF MSGFLG = 00010000 then
          Output RECEIVED_TRANSPORT_DATA_PACKET
     ELSEIF MSGFLG = 00010100 or 00000100 then
          Output RECEIVED_TRANSPORT_ACKNOWLEDGE_PACKET
     ELSEIF MSGFLG = 00011000, 00101000, 00111000, or 01001000 then
          Output RECEIVED_TRANSPORT_CONTROL_PACKET
     ELSEIF MSGFLG = 0101100 with a STARTTYPE of 1 or 2 then
          Output RECEIVED_TRANSPORT_INITIALIZATION_PACKET
     ELSE Null


PROCESS NAME:          DECODE DIALOGUE MESSAGE
PROCESS NUMBER:        1.3.1.1
PROCESS DESCRIPTION:
IF Input = command format then
     Output appropriate command:
          ENTER_MAINTENANCE_MODE
          CONNECT_REQUEST
          OPERATOR_PASSWORD_COMMAND
          OPERATOR_START_COMMAND
          ABORT_COMMAND
          DISCONNECT_REQUEST
ELSE output input as INCOMING_DIALOGUE_PACKET


PROCESS NAME:          EXAMINE ADJACENT NODE PARAMETERS AND DECODE
PROCESS NUMBER:        1.3.1.2
PROCESS DESCRIPTION:
IF ADJACENT_NODE_PARAMETER is equal to LINK_INACCESSABLE then
```

```
     Output LINK_INACCESSABLE_ERROR
ELSEIF Input = dialogue data then
     Output INCOMING_DIALOGUE_DATA
ELSE output INCOMING_DIALOGUE_INTERRUPT
```

```
PROCESS NAME:          BREAK DIALOGUE DATA INTO SEGMENTS
PROCESS NUMBER:        1.3.1.3
PROCESS DESCRIPTION:
TAKE INCOMING_DIALOGUE_DATA and break into segments according to the maximum
     segment length the node can handle.
     Output each segment as INCOMING_DIALOGUE_SEGMENT.
     Output the total number of segments the message has been broken up in
          as the SEGMENT_COUNT
```

```
PROCESS NAME:          EXAMINE DATA FLOW CONTROL PARAMETERS
PROCESS NUMBER:        1.3.1.4
PROCESS DESCRIPTION:
IF FLOW_CONTROL_PARAMETERS is set to closed then
     Output DATA_FLOW_ERROR
ELSEIF FLOW_CONTROL_PARAMETERS DATA_REQUEST_COUNT is greater than 0 then
     Decrement the DATA_REQUEST_COUNT and
     Output  the INCOMING_DIALOGUE_SEGMENT as TRANSPORT_DATA_SEGMENT
ELSE output DATA_FLOW_ERROR
```

```
PROCESS NAME:          PIGGYBACKED DATA ACKNOWLEDGE
PROCESS NUMBER:        1.3.1.5
PROCESS DESCRIPTION:
IF Input = DATA_ACK_PACKET or DATA_NAK_PACKET then
     Output TRANSPORT_DATA_SEGMENT with the appropriate ACKNUM field set
ELSE output PIGGYBACKED_TRANSPORT_DATA_SEGMENT with a null ACKNUM field
```

```
PROCESS NAME:          ASSIGN SEGMENT NUMBER MOD 4096
PROCESS NUMBER:        1.3.1.6
PROCESS DESCRIPTION:
IF TRANSMIT_D_FILE < 4096 then
          TRANSMIT_D_FILE = TRANSMIT_D_FILE + 1
          DATA_NAK_FILE = TRANSMIT_D_FILE + 1
     Output COUNTED_TRANSPORT_DATA_SEGMENT
     IF last segment then
          Output TRANSMIT_D_FILE value to LS_FILE
     ELSE Null
ELSE      TRANSMIT_D_FILE = 0
          DATA_NAK_FILE = 1
```

```
     Output COUNTED_TRANSPORT_DATA_SEGMENT
     IF last segment then
          Output TRANSMIT_D_FILE value to LS_FILE
     ELSE Null


PROCESS NAME:          LOAD AND DELETE DATA MEMORY
PROCESS NUMBER:        1.3.1.7
PROCESS DESCRIPTION:
IF Input = COUNTED_TRANSPORT_DATA_SEGMENT then
     Output to memory
ELSEIF Input = ACK_D_FILE value then
     Delete all COUNTED_TRANSPORT_DATA_SEGMENTS that have the same or lower
          values for segment numbers
ELSE Input = START_DATA flag then
     Output all recorded COUNTED_TRANSPORT_DATA_SEGMENTS with segment numbers
          equal to or greater than the DATA_NAK_FILE value as RETRANSMITTED_
          DATA_PACKET.  All COUNTED_TRANSPORT_DATA_SEGMENTS with segment
          numbers less than the DATA_NAK_FILE value are deleted.


PROCESS NAME:          CHECK DATA RETRANSMIT
PROCESS NUMBER:        1.3.1.8
PROCESS DESCRIPTION:
IF DATA_NAK_FILE = TRANSMIT_D_FILE + 1 then
     Output COUNTED_TRANSPORT_DATA_PACKET as TRANSMITTED_DATA_PACKET
ELSE the DATA_NAK_FILE value will be substracted from the TRANSMIT_I/L_FILE
     value and this value will be used to increment the
FLOW_CONTROL_PARAMETERS DATA_REQUEST_COUNT value.
     Output START_DATA flag
     Buffer all incoming COUNTED_TRANSPORT_DATA_SEGMENTS


PROCESS NAME:          CODE DATA AND I/L PACKETS
PROCESS NUMBER:        1.3.1.9
PROCESS DESCRIPTION:
IF Input = RETRANSMITTED_DATA_PACKETS, TRANSMITTED_DATA_PACKETS,
     RETRANSMITTED_I/L_PACKET, or TRANSMITTED_I/L_PACKETS then
         Output as TRANSMITTED_TRANSPORT_DATA_PACKET
ELSE Null


PROCESS NAME:          EXAMINE INTERRUPT FLOW CONTROL PARAMETERS
PROCESS NUMBER:        1.3.2.1
PROCESS DESCRIPTION:
IF FLOW_CONTROL_PARAMETER INTERRUPT_REQUEST_COUNT is greater than 0 then
     Decrement the INTERRUPT_REQUEST_COUNT and
```

```
     Output the INCOMING_DIALOGUE_INTERRUPT as TRANSPORT_INTERRUPT_PACKET
ELSE output INTERRUPT_FLOW_ERROR


PROCESS NAME:          GENERATE INTERRUPT LINK SERVICES
PROCESS NUMBER:        1.3.2.2
PROCESS DESCRIPTION:
UPON reception of the first INCOMING_DIALOGUE_INTERRUPT
     Generate and output an INTERRUPT_LINK_SERVICES_PACKET with LSFLAGS field
     Set to INTERRUPT_REQUEST_COUNT plus a START_DATA command.  The actual
     INTERRUPT_DATA_COUNT is contained in the FCVAL field.


PROCESS NAME:          PIGGYBACK I/L ACKNOWLEDGE
PROCESS NUMBER:        1.3.2.3
PROCESS DESCRIPTION:
IF Input = I/L_ACK_PACKET or I/L_NAK_PACKET then
     Output TRANSPORT_INTERRUPT_PACKET with the apporpriate ACKNUM field set
ELSE output PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET with a null ACKNUM field


PROCESS NAME:          CODE I/L PACKET
PROCESS NUMBER:        1.3.2.4
PROCESS DESCRIPTION:
IF Input = PIGGYBACKED_TRANSPORT_INTERRUPT_PACKET, INTERRUPT_LINK_SERVICES_
     PACKET, or DATA_LINK_SERVICES_PACKET then
          Output as TRANSPORT_I/L_PACKET
ELSE Null


PROCESS NAME:          GENERATE DATA LINK SERVICES
PROCESS NUMBER:        1.3.2.5
PROCESS DESCRIPTION:
IF Input = SEGMENT_COUNT then
     Output a DATA_LINK_SERVICES_PACKET with LSFLAGS field set to
     DATA_REQUEST_COUNT
     Output a START_DATA command. The actual DATA_REQUEST_COUNT value is
     contained in the FCVAL field
ELSEIF Input = LS_CODE then
     Output a DATA_LINK_SERVICES_PACKET with LSFLAG field set to DATA_REQUEST_
          COUNT.
     Output a STOP_DATA command. The actual DATA_REQUEST_COUNT value is the
          amount of segment space left over from filling the last buffer.
ELSE Null
```

```
PROCESS NAME:          ASSIGN PACKET NUMBER MOD 4096
PROCESS NUMBER:        1.3.2.6
PROCESS DESCRIPTION:
IF TRANSMIT_I/L_FILE < 4096 then
     TRANSMIT_I/L_FILE = TRANSMIT_I/L_FILE + 1
     I/L_NAK_FILE = TRANSMIT_I/L_FILE + 1
     Output COUNTED_TRANSPORT_I/L_PACKET
ELSE TRANSMIT_I/L_FILE = 0
     I/L_NAK_FILE = 1
     Output COUNTED_TRANSPORT_I/L_PACKET



PROCESS NAME:          LOAD AND DELETE I/L MEMORY
PROCESS NUMBER:        1.3.2.7
PROCESS DESCRIPTION:
IF Input = COUNTED_TRANSPORT_I/L_PACKET then
     Load to MEMORY
ELSEIF Input = ACK_I/L_FILE value then
     Delete all COUNTED_TRANSPORT_I/L_PACKETS that have the same of lower
          values for segment numbers
ELSE Input = START_I/L flag then
     Output all recorded COUNTED_TRANSPORT_I/L_PACKETS with segment numbers
          equal to or greater than the I/L_NAK_FILE value as RETRANSMITTED_
          I/L_PACKETS.
     Delete all COUNTED_TRANSPORT_I/L_PACKETS with segment numbers less than
          the I/L_NAK_FILE value



PROCESS NAME:          CHECK I/L RETRANSMIT
PROCESS NUMBER:        1.3.2.8
PROCESS DESCRIPTION:
IF I/L_NAK_FILE = TRANSMIT_I/L_FILE + 1 then
     Output COUNTED_TRANSPORT_I/L_PACKET as TRANSMITTED_I/L_PACKET
ELSE output START_I/L flag and
     Buffer incoming COUNTED_TRANSPORT_I/L_PACKET



PROCESS NAME:          DETERMINE DATA PACKET TYPE
PROCESS NUMBER:        1.4.1
PROCESS DESCRIPTION:
IF Input MSGFLG = 0BB00000 where BB = 00,01,10,11 then
     Output RECEIVED_NORMAL_DATA_SEGMENT
ELSEIF Input MSGFLG = 00110000 or 00010000 then
     Output RECEIVED_I/L_PACKET
ELSE Null
```

```
PROCESS NAME:          DECODE DATA ACKNUM
PROCESS NUMBER:        1.4.2
PROCESS DESCRIPTION:
IF RECEIVED_NORMAL_DATA_SEGMENT ACKNUM field = NAK then
     Output data segment number to DATA_NAK_FILE
ELSEIF RECEIVED_NORMAL_DATA_SEGMENT ACKNUM field = ACK then
     IF RECEIVED_NORMAL_DATA_SEGMENT ACKNUM value is greater than the stored
          ACK_D_FILE value then
          Output segment number to the ACK_D_FILE
     ELSE Null
ELSE Null


PROCESS NAME:          DECODE DATA SEGMENT NUMBER
PROCESS NUMBER:        1.4.3
PROCESS DESCRIPTION:
IF Input data segment number is greater than the RECEIVED_D_FILE value then
     Update the RECEIVED_D_FILE value then
     IF the FLOW_CONTROL_PARAMETER DATA_FLOW_CONTROL_SWITCH is set to open
then
          Output VALID_NORMAL_DATA_SEGMENT and DATA_ACK_FLAG
     ELSE output CLOSED_FLOW_CONTROL
ELSE output DATA_NAK_FLAG


PROCESS NAME:          DECODE I/L SEGMENT NUMBER
PROCESS NUMBER:        1.4.4
PROCESS DESCRIPTION:
IF Input I/L segment number is greater than the RECEIVED_I/L_FILE value then
     Update the RECEIVED_I/L_FILE and
     Output I/L_ACK_FLAG and VALID_I/L_PACKET
ELSE output I/L_NAK_FLAG


PROCESS NAME:          DECODE I/L ACKNUMI FIELD
PROCESS NUMBER:        1.4.5
PROCESS DESCRIPTION:
IF RECEIVED_I/L_PACKET ACKNUMI field is equal to NAK then
     Output stored I/L_SEGMENT_NUMBER to I/L_NAK_FILE
ELSEIF RECEIVED_I/L_PACKET ACKNUMI field is greated than the stored ACK_I/L_
     FILE value then
     Output segment number to the ACK_I/L_FILE
ELSE Null


PROCESS NAME:          GENERATE DATA ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.6
```

```
PROCESS DESCRIPTION:
UPON reception of a DATA_ACK_FLAG
     Generate and output a DATA_ACK_PACKET


PROCESS NAME:          GENERATE DATA NEGATIVE ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.7
PROCESS DESCRIPTION:
UPON reception of a DATA_NAK_FLAG or CLOSED_FLOW_CONTROL flag
     Output a DATA_NAK_PACKET


PROCESS NAME:          LOAD RECEIVE BUFFER UNTIL FULL OR LS
PROCESS NUMBER:        1.4.8
PROCESS DESCRIPTION:
IF receive buffers available then
     Fill until all data segments are accounted for and
     Output NORMAL_DATA_PACKET
     IF last receive buffer is not entirely full when last data segment is
          deposited then
          Output LS_CODE
     ELSE Null
ELSE output NO_BUFFER_SPACE


PROCESS NAME:          GENERATE I/L NEGATIVE ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.9
PROCESS DESCRIPTION:
UPON reception of an I/L_NAK_FLAG
     Generate and output an I/L_NAK_PACKET


PROCESS NAME:          GENERATE I/L ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.10
PROCESS DESCRIPTION:
UPON reception of an I/L_ACK_FLAG
     Generate and output an I/L_ACK_PACKET


PROCESS NAME:          DECODE VALID I/L PACKET
PROCESS NUMBER:        1.4.11
PROCESS DESCRIPTION:
IF Input MSGFLG = 00010000 then
     Output VALID_LINK_SERVICES_PACKET
ELSEIF Input MSGFLG = 00110000 then
     IF INTERRUPT_REQUEST_COUNT is greater than 0 then
```

```
          Output VALID_INTERRUPT_PACKET
     ELSE output INTERRUPT_ERROR
ELSE Null
```

```
PROCESS NAME:          CODE ACKNOWLEDGE PACKETS
PROCESS NUMBER:        1.4.12
PROCESS DESCRIPTION:
IF Input = DATA_ACK_PACKET, DATA_NAK_PACKET, I/L_ACK_PACKET, or I/L_NAK_PACKET
     then
     Output as TRANSMITTED_TRANSPORT_ACKNOWLEDGE_PACKET
ELSE Null
```

```
PROCESS NAME:          CODE DATA PACKETS
PROCESS NUMBER:        1.4.13
PROCESS DESCRIPTION:
IF Input = NORMAL_DATA_PACKET or VALID_INTERRUPT_PACKET then
     Output as OUTGOING_DIALOGUE_DATA_PACKET
ELSE Null
```

```
PROCESS NAME:          DECODE LINK SERVICES PACKET
PROCESS NUMBER:        1.4.14
PROCESS DESCRIPTION:
IF FCVAL field value of the VALID_LINK_SERVICES_PACKET when added to the data
     segment request would result in a DATA_SEGMENT_REQUEST_COUNT greater than
     +127 or less than -127 then
     Output LINK_SERVICES_ERROR
ELSEIF FCVAL field value of the VALID_LINK_SERVICES_PACKET when added to the
     I/L segment request would result in a INTERRUPT_REQUEST_COUNT greater
     than +127 then
     Output LINK_SERVICES_ERROR
ELSE output FCVAL field value + current SEGMENT_REQUEST_COUNT to the FLOW_
     CONTROL_PARAMETERS_TABLE
```

```
PROCESS NAME:          TRANSITION LINK TO ON-LINE MODE
PROCESS NUMBER:        1.5.1
PROCESS DESCRIPTION:
IF Input = OPERATOR_START_COMMAND then
     Output INITIALIZE_LINK command
ELSE Input = INITIALIZATION_ON_OTHER_END or INITIALIZATION_COMPLETE then
     Output START_COMMAND
```

```
PROCESS NAME:          DECODE FUNCTIONS FIELD
PROCESS NUMBER:        1.5.2
PROCESS DESCRIPTION:
IF Input = RECEIVED_NODE_INITIALIZATION_PACKET with FUNCTIONS field set to
     INTERCEPT then
     Output RECEIVED_INTERUPT_INITIALIZATION_PACKET
ELSE Input = RECEIVED_NODE_INITIALIZATION_PACKET with FUNCTIONS field set to
     NO_INTERCEPT then
     Output RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET


PROCESS NAME:          DECODE INITIALIZATION PACKET
PROCESS NUMBER:        1.5.3
PROCESS DESCRIPTION:
IF Input STARTTYPE = 1 then
     Output RECEIVED_NODE_INITIALIZATION_PACKET
ELSEIF Input STARTTYPE = 2 then
     Output RECEIVED_NODE_VERFICATION_PACKET
ELSEIF Input STARTTYPE = 4 then
     Output RECEIVED_CORRECT_PASSWORD_COMMAND
ELSEIF Input STARTTYPE = 5 then
     Output RECEIVED_INCORRECT_PASSWORD_COMMAND
ELSE Input STARTTYPE = 3 then Null


PROCESS NAME:          GENERATE NODE INITIALIZATION PACKET VERIFY = 0
PROCESS NUMBER:        1.5.4
PROCESS DESCRIPTION:
IF Input = START_COMMAND or VERIFY_CLEAR flag then
     Output NODE_INITIALIZATION_VO_PACKET
ELSE Null


PROCESS NAME:          DECODE REMAINING FIELDS
PROCESS NUMBER:        1.5.5
PROCESS DESCRIPTION:
IF Input = RECEIVED_INTERCEPT_INITIALIZATION_PACKET then
     Send required field values to INITIAL_PARAMETER_LIST
     IF VERIFY field = 1 then
          Output VERIFY_SET flag
     ELSE output VERIFY_CLEAR flag
ELSE Null


PROCESS NAME:          GENERATE NODE VERIFICATION PACKET
PROCESS NUMBER:        1.5.6
PROCESS DESCRIPTION:
```

```
IF Input = VERIFY_SET then
     Take OPERATOR_PASSWORD_COMMAND and
     Generate and output NODE_VERIFICATION_PACKET
ELSE Null



PROCESS NAME:          GENERATE NODE INITIALIZATION PACKET VERIFY = 1
PROCESS NUMBER:        1.5.7
PROCESS DESCRIPTION:
IF Input = RECEIVED_NO_INTERCEPT_INITIALIZATION_PACKET then
     Decode fields and place values in INITIAL_PARAMETER_LIST and
     Generate and output NODE_INITIALIZATION_V1_PACKET
ELSE Null



PROCESS NAME:          DECODE PASSWORD
PROCESS NUMBER:        1.5.8
PROCESS DESCRIPTION:
IF PASSWORD field of a RECEIVED_NODE_VERIFICATION_PACKET contains the correct
     password then
     Output the password to the SATELLITE_NODE_PARAMETERS_TABLE and
     Output CORRECT_PASSWORD_COMMAND
ELSE output INCORRECT_PASSWORD_COMMAND



PROCESS NAME:          CODE INITIALIZATION PACKETS
PROCESS NUMBER:        1.5.9
PROCESS DESCRIPTION:
IF Input = NODE_INITIALIZATION_V0_PACKET, NODE_VERIFICATION_PACKET, NODE_
     INITIALIZATION_V1_PACKET, CORRECT_PASSWORD_COMMAND, or INCORRECT_
     PASSWORD_COMMAND then
     Output as TRANSMITTED_TRANSPORT_INITIALIZATION_PACKET
ELSE Null



PROCESS NAME:          GENERATE CONNECT INITIATE PACKET
PROCESS NUMBER:        1.6.1.1
PROCESS DESCRIPTION:
IF Input = CONNECT_REQUEST command then
     Output CONNECT_INITIATE_PACKET
ELSE Null



PROCESS NAME:          DECODE CONTROL PACKET
PROCESS NUMBER:        1.6.1.2
PROCESS DESCRIPTION:
```

```
IF Input MSGFLG = 00011000 then
     Output RECEIVED_CONNECT_INITIATE_PACKET
ELSEIF Input MSGFLG = 00101000 then
     Output RECEIVED_CONNECT_CONFIRM_PACKET
ELSEIF Input MSGFLG = 00111000 then
     Output RECEIVED_DISCONNECT_INITIATE_PACKET
ELSE Input MSGFLG = 01001000 then
     Output RECEIVED_DISCONNECT_CONFIRM_PACKET



PROCESS NAME:          DECODE CONNECT INITIATE PACKET
PROCESS NUMBER:        1.6.1.3
PROCESS DESCRIPTION:
PASS SOURCE_NAME, SOURCE_PROCESS_IDENTIFICATION, DESTINATION_PROCESS_
     IDENTIFICATION, REPLY_IDENTIFIERS, ACCESS_CONTROL_INFORMATION, and
     REMOTE_PROCESS'S_SEGMENT_SIZE to the DIALOGUE_PROCESS_TABLE
PASS SERVICES to the FLOW_CONTROL_PARAMETERS_TABLE
IF the following errors apply then
     Output a REJECT_CONNECT flag
          1.  Resource allocation failure
          2.  Destination node does not exist
          3.  Node shutting down
          4.  Destination process does not exist
          5.  Invalid process name field
          6.  Destination process queue overflow
          7.  Too many connections to node
          8.  Too many connections to destination node
          9.  Access not permitted-unacceptable RQSTRID or PASSWORD
         10.  Unacceptable ACCOUNT information
         11.  Dialogue process aborted, timedout, or cancelled request
         12.  No path to destination node
         13.  Image data field too long
ELSE output an ACCEPT_CONNECT flag



PROCESS NAME:          DIALOGUE PROCESS EXTERNAL END
PROCESS NUMBER:        1.6.1.4
PROCESS DESCRIPTION:
IF Input = DISCONNECT_REQUEST then
     IF ACK_D_FILE value equals the LS_FILE value then
          Output a STOP flag
     ELSE hold DISCONNECT_REQUEST
ELSE Null



PROCESS NAME:          DECODE CONNECT CONFIRM PACKET
PROCESS NUMBER:        1.6.1.5
PROCESS DESCRIPTION:
```

```
IF the following errors apply then
     Output a REJECT_CONFIRM flag
          1.  Segsize too small
          2.  Dialogue process aborted, timedout, or cancelled
          3.  DSTADDR logical link does not exist
          4.  Image data field too long
ELSE output an ACCEPT_CONFIRM flag
     Update the FLOW_CONTROL_PARAMETERS_TABLE
     Signal ADJACENT_NODE_PARAMETERS that values are valid


PROCESS NAME:          GENERATE CONNECT CONFIRM PACKET
PROCESS NUMBER:        1.6.1.6
PROCESS DESCRIPTION:
UPON successful reception of an ACCEPT_CONNECT flag
     Generate and output a CONNECT_CONFIRM_PACKET using data in the DIALOGUE_
          PROCESS_TABLE


PROCESS NAME:          GENERATE DISCONNECT INITIATE PACKET
PROCESS NUMBER:        1.6.1.7
PROCESS DESCRIPTION:
IF Input = REJECT_CONNECT then
     Output a DISCONNECT_INITIATE_PACKET with REASON as listed in process
          1.6.1.3
ELSEIF Input = REJECT_CONFIRM then
     Output a DISCONNECT_INITIATE_PACKET with REASON as listed in process
          1.6.1.5
ELSEIF Input = PERSISTENT_ERROR then
     Output a DISCONNECT_INITIATE_PACKET reason: "Third party aborted the
     logical link"
ELSEIF Input = STOP then
     Output a DISCONNECT_INITIATE_PACKET with reason: "Third party aborted
     the logical link"
ELSEIF Input = ABORT_COMMAND then
     Output a DISCONNECT_INITIATE_PACKET with reason: "Link aborted by
     dialogue process"
ELSEIF Input = INTERRUPT_ERROR then
     Output a DISCONNECT_INITIATE_PACKET with reason as listed in REASON table
ELSE Input = LINK_SERVICES_ERROR then
     Output a DISCONNECT_INITIATE_PACKET with reason as listed in REASON table


PROCESS NAME:          CODE CONTROL PACKETS
PROCESS NUMBER:        1.6.1.8
PROCESS DESCRIPTION:
IF Input = CONNECT_INITIATE_PACKET, CONNECT_CONFIRM_PACKET,
     DISCONNECT_INITIATE_PACKET, or DISCONNECT_CONFIRM_PACKET then
```

```
     Output as TRANSMITTED_TRANSPORT_CONTROL_PACKET
ELSE Null


PROCESS NAME:          DECODE DISCONNECT CONFIRM PACKET
PROCESS NUMBER:        1.6.2.1
PROCESS DESCRIPTION:
DECODE REASON field and
     PASS to DIALOGUE_PROCESS
     SHUT-DOWN the logical link


PROCESS NAME:          DECODE DISCONNECT INITIATE PACKET
PROCESS NUMBER:        1.6.2.2
PROCESS DESCRIPTION:
IF REASON field = "link aborted by dialogue process" then
     Output REASON to DIALOGUE_PROCESS and
     Output an ABORT flag
ELSE Pass error reason to DIALOGUE_PROCESS and
     Output a NO_ABORT flag


PROCESS NAME:          DIALOGUE PROCESS END
PROCESS NUMBER:        1.6.2.3
PROCESS DESCRIPTION:
IF Input = NO_ABORT then
     IF ACK_D_FILE value = LS_FILE value then
          Output STOP_LINK_NOW flag
     ELSE hold NO_ABORT flag
ELSE Null


PROCESS NAME:          GENERATE DISCONNECT CONFIRM PACKET
PROCESS NUMBER:        1.6.2.4
PROCESS DESCRIPTION:
IF Input = ABORT flag or STOP_LINK_NOW flag then
     Output a DISCONNECT_CONFIRM_PACKET with reason: "Confirmation of
     disconnect initiate"
     Output a STOP_LINK command to shut down the physical link
ELSE Null


PROCESS NAME:          DETERMINE ACKNOWLEDGE TYPE
PROCESS NUMBER:        1.7.1
PROCESS DESCRIPTION:
IF Input MSGFLG = 00000100 then
```

```
     Output as RECEIVED_NORMAL_DATA_ACK_PACKETS
ELSE Input MSGFLG = 00010100 then
     Output as RECEIVED_I/L_ACK_PACKET


PROCESS NAME:          DECODE DATA ACK PACKET
PROCESS NUMBER:        1.7.2
PROCESS DESCRIPTION:
DECODE ACKNUM field
     IF = 1 then
          Output associated value to ACK_D_FILE
     ELSE = 2 then
          Output associated value to DATA_NAK_FILE


PROCESS NAME:          DECODE_I/L_ACK_PACKET
PROCESS NUMBER:        1.7.3
PROCESS DESCRIPTION:
DECODE ACKNUMI field
     IF = 1 then
          Output associated value to the ACK_I/L_FILE
     ELSE = 2 then
          Output associated value to the I/L_NAK_FILE


PROCESS NAME:          EXAMINE DSTADDR FIELD
PROCESS NUMBER:        1.8.1
PROCESS DESCRIPTION:
IF DSTADDR = ADJACENT_NODE_ADDRESS then
     Output as ADJACENT_NODE_PACKETS
ELSEIF INITIALIZATION_PACKETS then
     Output as ADJACENT_NODE_PACKETS
ELSEIF DSTADDR = NONADJACENT_NODE_ADDRESS then
     Output as NONADJACENT_NODE_PACKETS
ELSE Input = CONNECT_INITIATE_PACKET then
     Output as NONADJACENT_NODE_PACKETS


PROCESS NAME:          ADD RTHDR FIELD
PROCESS NUMBER:        1.8.2
PROCESS DESCRIPTION:
IF Input = NONADJACENT_NODE_PACKETS then
     Refer to DIALOGUE_PROCESS_TABLE and
     Extract DSTNODE_NAME
     Output RTHDR_NONADJACENT_NODE_PACKETS
ELSE Null
```

```
PROCESS NAME:          IS ROUTING NECESSARY AND PRESENT
PROCESS NUMBER:        1.8.3
PROCESS DESCRIPTION:
IF Input DSTNODE name = the name of a satellite not adjacent to this node then
     Output as NONADJACENT_ROUTE_PACKET
ELSE output as NONADJACENT_SATELLITE_PACKETS


PROCESS NAME:          PASS TO CORRECT ADJACENT NODE
PROCESS NUMBER:        1.8.4
PROCESS DESCRIPTION:
IF Input = ADJACENT_NODE_PACKETS or NONADJACENT_SATELLITE_PACKETS then
     Output as OUTGOING_SATELLITE_TRANSPORT_PACKET
     Output a TRANSMIT_PACKET command
     Output a RECEIVE_PACKET command
ELSE Null


PROCESS NAME:          CODE OUTGOING DIALOGUE MESSAGE
PROCESS NUMBER:        1.9
PROCESS DESCRIPTION:
IF Input = FLOW_CONTROL_ERRORS, OUTGOING_DIALOGUE_DATA_PACKET, RECEIVED_
     INCORRECT_PASSWORD_COMMAND, ERROR_REASON, or TRANSIENT_ERROR_THRESHOLD_
     COUNTER_OVERFLOW then
     Output above as OUTGOING_DIALOGUE_MESSAGE
ELSE Null


PROCESS NAME:          SEND TO NETWORK PROTOCOL
PROCESS NUMBER:        1.10
PROCESS DESCRIPTION:
IF Input = OUTGOING_TRANSPORT_ROUTE_PACKET or NONADJACENT_ROUTE_PACKET then
     Output OUTGOING_NODE_TRANSPORT_PACKET
ELSE Null


PROCESS NAME:          DECODE MESSAGE TYPE
PROCESS NUMBER:        2.1
PROCESS DESCRIPTION:
IF Input = NETWORK_HEADER then
     Output as NETWORK_PACKET
ELSE Input = ROUTE_HEADER then
     Output as a ROUTING_PACKET
```

```
PROCESS NAME:          CHECK R-HOP COUNT = 2
PROCESS NUMBER:        2.2
PROCESS DESCRIPTION:
IF Input HOP_COUNT field is less than or equal to 2 then
     Output VALID_ROUTING_PACKET
ELSE Null


PROCESS NAME:          DECODE NETWORK HEADER
PROCESS NUMBER:        2.3
PROCESS DESCRIPTION:
IF Input NETWORK layer header destination field value relates to an adjacent
     satellite node then
     Output as INCOMING_NODE_TRANSPORT_PACKET after stripping off the network
          header
ELSE output as NETWORK_TO_NETWORK_PACKET


PROCESS NAME:          INCREMENT HOP
PROCESS NUMBER:        2.4
PROCESS DESCRIPTION:
INCREMENT value in HOP COUNT field by 1 and
     Output as HOPPED_ROUTING_PACKET


PROCESS NAME:          UPDATE ROUTING TABLE IF NEW
PROCESS NUMBER:        2.5
PROCESS DESCRIPTION:
IF VALID_ROUTING_PACKET line cost value is different from what is located in
     ROUTING_TABLE_2 link cost value for the particular link then
     Update to the correct ROUTING_TABLE_2 value
     Calculate and update 1-2-3, 2-3-1, and 3-1-2 values if possible
ELSE Null


PROCESS NAME:          DETERMINE LEAST COST LINK
PROCESS NUMBER:        2.6
PROCESS DESCRIPTION:
FROM the OUTGOING_NODE_TRANSPORT_PACKET RTHDR field use the NODENAME to
     Determine the least cost link
     Output TRANSPORT_PACKET
ELSE Null


PROCESS NAME:          ADD NETWORK HEADER
```

```
PROCESS NUMBER:        2.7
PROCESS DESCRIPTION:
IF Input = TRANSPORT_PACKET then
     Add network layer header and
     Send out as TRANSPORT_TO_NETWORK_PACKET. HOP = 1
ELSE Null



PROCESS NAME:          SEND OLD VALUES OUT OVER NEW INITIALIZED LINE
PROCESS NUMBER:        2.8
PROCESS DESCRIPTION:
IF node senses that initialization just took place over a new routing line
then
     IF ROUTING_TABLE_2 has existing values then
          Generate and output OLD_ROUTING_PACKETS
     ELSE Null
ELSE Null



PROCESS NAME:          CHECK N-HOP COUNT = 2
PROCESS NUMBER:        2.9
PROCESS DESCRIPTION:
IF Input HOPS field is less than or equal to 2 then

     Output as VALID_NETWORK_TO_NETWORK_PACKET
ELSE Null



PROCESS NAME:          UPDATE NETWORK HEADER
PROCESS NUMBER:        2.10
PROCESS DESCRIPTION:
IF Input = VALID_NETWORK_TO_NETWORK_PACKET then
     Compare existing destination code in the network header against the line
          cost in ROUTING_TABLE_2 then
          Output over the correct ROUTING_LINE
ELSE Null



PROCESS NAME:          EVERY TIMER INTERVAL UPDATE ADJACENT LINE
PROCESS NUMBER:        2.11
PROCESS DESCRIPTION:
AT the end of every preset timer interval
     check ROUTING_TABLE_2
          IF values  present then
               Output ADJACENT_NODE_ROUTING_PACKETS on all routing links.
               HOP = 1
          ELSE Null
```

```
PROCESS NAME:          ISSUE CORRECT LINK COST OVER ALL LINKS
PROCESS NUMBER:        2.12
PROCESS DESCRIPTION:
IF Input = STOP_LINK command or INITIALIZE_LINK command then refer to ROUTING_
     TABLE_1 and get the cost associated with the specific link.
     Generate and output INITIAL_ROUTE_PACKET
ELSE Null


PROCESS NAME:          SEND PACKET OVER CORRECT LINE(S)
PROCESS NUMBER:        2.13
PROCESS DESCRIPTION:
IF Input = INITIAL_ROUTE_PACKET, VALID_NETWORK_PACKET, HOPPED_ROUTIN_ _PACKET,
     ADJACENT_NODE_ROUTING_PACKET, OLD_ROUTING_PACKETS, or TRANSPORT_TO_
     NETWORK_PACKET then
     Output it as a OUTGOING_NODE_NETWORK_PACKET
ELSE Null


PROCESS NAME:          EXECUTE NETWORK PROTOCOL AT SECONDARY NODE
PROCESS NUMBER:        3.0
PROCESS DESCRIPTION:
USE the established ROUTING ALGORITHM to route packets from SECONDARY_INCOMING
     _NODE_NETWORK_PACKET to SECONDARY_INCOMING_NODE_TRANSPORT_PACKET or
     SECONDARY_OUTGOING_NODE_NETWORK_PACKET.  Also route packets from
     SECONDARY_OUTGOING_TRANSPORT_PACKET to SECONDARY_OUTGOING_NODE_NETWORK_
     PACKET.  Routing packets are generated internally to keep all routing
     tables up-to-date.


PROCESS NAME:          EXECUTE TRANSPORT PROTOCOL AT SECONDARY NODE
PROCESS NUMBER:        4.0
PROCESS DESCRIPTION:
USED to take a SECONDARY_INCOMING_DIALOGUE_MESSAGE or SECONDARY_INCOMING_
     NODE_TRANSPORT_PACKET or SECONDARY_INCOMING_SATELLITE_TRANSPORT_PACKET
     and create the necessary envelope for accurate interprocess communication
     Outputs can be SECONDARY_OUTGOING_TRANSPORT_PACKET or SECONDARY_OUTGOING_
     NODE_TRANSPORT_PACKET or SECONDARY_OUTGOING_DIALOGUE_MESSAGE.
```

DATA DICTIONARY

FOR LINK LEVEL (L) PROTOCOL

Contents

DATA FLOW NAME: ABUTTED_PACKET
ALIASES: FOLLOWON_PACKET
SYNC_PACKET
LINK_PACKETS
COMPOSITION:
SEE ALIASES
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA FLOW NAME: ACK_PACKET
ALIASES: RECEIVED_ACK_PACKET
COMPOSITION:
ACK_PACKET = ENQ + ACKTYPE + ACKSUB + FLAGS + RESP +
FILL + ADDR + BLKCK3
NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKET LAYER

DATA ELEMENT NAME: ACKSUB
ALIASES: NONE
VALUES AND MEANINGS:
ACK SUBTYPE = 00000000
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: ACKTYPE
ALIASES: NONE
VALUES AND MEANINGS:
ACK PACKET TYPE VALUE = 00000001
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: ADDR
ALIASES: NONE
VALUES AND MEANINGS:
ALWAYS EQUALS 1
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: AGREE
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO ISSUE AN ACK_PACKET SINCE THE LAST
SEQUENTIAL NUMBERED DATA PACKET SENT AGREES WITH THE
NUMBER OF THE LAST SEQUENTIAL PACKET RECEIVED.
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: BLKCK1
ALIASES: NONE
VALUES AND MEANINGS:

THE BLOCK CHECK ON THE NUMBERED PACKET HEADER. IT IS COMPUTED ON SOH THROUGH ADDR USING THE CRC-16 POLYNOMIAL ($X^{-}16 + X^{-}15 + X^{-}2 + 1$). BLKCK1 IS INITIALIZED TO ZERO PRIOR TO COMPUTATION AND TRANSMITTED X 15 BIT FIRST. ON RECEPTION THE INCLUSION OF BLKCK1 IN THE COMPUTATION WILL RESULT IN A ZERO REMAINDER OR CRC IF NO ERRORS EXIST.

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: BLKCK2
ALIASES: NONE
VALUES AND MEANINGS:

THE BLOCK CHECK ON THE DATA FIELD. COMPUTED ON THE DATA FIELD ONLY USING THE TECHNIQUE DESCRIBED FOR BLKCK1.

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: BLKCK3
ALIASES: NONE
VALUES AND MEANINGS:

THE BLOCK CHECK ON THE CONTROL PACKET. BLKCK3 IS COMPUTED ON FIELDS ENQ THROUGH ADDR USING THE TECHNIQUE DESCRIBED IN BLKCK1.

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: CLEAR
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG USED TO CLEAR THE SNAK FLAG FILE OR THE SACK FLAG FILE OR THE SREP FLAG FILE

NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: COUNT
ALIASES: NONE
VALUES AND MEANINGS:

SPECIFIES THE NUMBER OF BYTES IN THE DATA FIELD. A VALUE OF 0 IS NOT ALLOWED.

NOTES: ALL LINK LAYERS

DATA FLOW NAME: COUNTED_LINK_DATA_PACKET
ALIASES: < 256_LINK_PACKET
RETRANSMITTED_PACKETS
NOSYNC_PRIMARY_TO_SECONDARY_DATA_PACKET
COMPOSITION:
COUNTED_LINK_DATA_PACKET = LINK_DATA_PACKET + RESP + NUM
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: DATA
ALIASES: NONE
VALUES AND MEANINGS:
THE NUMBERED PACKED DATA FIELD. THIS FIELD IS TOTALLY TRANSPARENT TO THE PROTOCOL AND HAS NO RESTRICTIONS EXCEPT TO CONTAIN THE NUMBER OF BYTES SPECIFIED IN THE COUNT FIELD.
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: DISABLE_LINK
ALIASES: NONE
VALUES AND MEANINGS:
DISCONNECTS THE DRIVER FROM THE LINK PROTOCOL.
NOTES: START REPLY TIMER LAYER

DATA ELEMENT NAME: DISAGREE
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG PLUS THE NUMBER OF THE LAST SEQUENTIAL PACKET THAT WAS RECEIVED. USED TO ISSUE A NAK_PACKET SINCE THE LAST SEQUENTIALLY NUMBERED DATA PACKET SENT DID NOT AGREE WITH THE LAST SEQUENTIAL PACKET RECEIVED.
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: DLE
ALIASES: NONE
VALUES AND MEANINGS:
MAINTENANCE PACKET IDENTIFIED = 10010101
NOTES: ALL LINK LAYERS

DATA FLOW NAME: DLE_PACKET
ALIASES: VALID_DLE_PACKET
TRANSMIT_DLE_PACKET
COMPOSITION:
DLE_PACKET = DLE + COUNT + FLAGS + FILL + FILL + ADDR +
BLKCK1 + DATA + BLKCK2
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER
FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE MAINTENANCE PACKET LAYER

DATA ELEMENT NAME: ENABLE_LINK
ALIASES: NONE
VALUES AND MEANINGS:
CONNECTS THE DRIVER TO THE LINK PROTOCOL
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: ENQ
ALIASES: NONE
VALUES AND MEANINGS:
UNNUMBERED CONTROL PACKET IDENTIFIER = 00000101
NOTES: ALL LINK LAYERS

DATA FLOW NAME: ENQ_PACKET
ALIASES: VALID_ENQ_PACKET
VALID_CRC_ENQ_PACKET
VALID_LENGTH_ENQ_HEADER
COMPOSITION:
ENQ_PACKET = ENQ + TYPE + SUBTYPE + FLAGS + RCVR +
SNDR + ADDR + BLKCK3
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER
FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA FLOW NAME: ENQ/DLE_SYNC_PACKET
ALIASES: QSYNC_ENQ/DLE_PACKET
COMPOSITION:
ENQ/DLE_SYNC_PACKET = 4{SYNC} + [TRANSMIT_DLE_PACKET
PAD/STRT_PACKET
MOP_PACKET
REP_PACKET
STACK_PACKET
ACK_PACKET
NAK_PACKET]

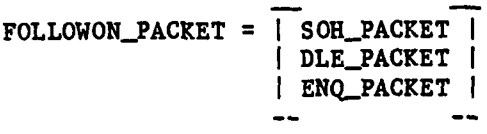
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: ENTER_MAINTENANCE_MODE
ALIASES: NONE
VALUES AND MEANINGS:
CHANGE FROM ON-LINE MODE TO OFF-LINE MAINTENANCE MODE
NOTES: EXECUTE MAINTENANCE PACKET LAYER

DATA ELEMENT NAME: FILL
ALIASES: NONE
VALUES AND MEANINGS:
FILL BYTE WITH VALUE OF 0
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: FLAGS
ALIASES: NONE
VALUES AND MEANINGS:
LINK FLAGS TO CONTROL OWNERSHIP AND PACKET SYNCHRONIZATION
BIT 0 = QUICK SYNC FLAG - QSYNC
BIT 1 = SELECT FLAG = OWNERSHIP (MUST BE PRESENT ON DLE AND ENQ PACKETS, DON'T CARE IN SOH_PACKET).
NOTES: ALL LINK LAYERS

DATA FLOW NAME: FOLLOWON_PACKET
ALIASES: SYNC_PACKET
ABUTTED_PACKET
LINK_PACKETS
COMPOSITION:



NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA FLOW NAME: FRAMING_ERRORS
ALIASES: NONE
COMPOSITION:

```
                                        | INVALID_DATA_LENGTH |
                                        | INVALID_DLE_PACKET  |
                                        | INVALID_BLKCK1      |
                                        | INVALID_BLKCK2      |
                                        | INVALID_BLKCK3      |
                                        --                    --
NOTES:                  EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER


DATA ELEMENT NAME:      HEADER_FORMAT_ERROR
ALIASES:                NONE
VALUES AND MEANINGS:
                        FLAGS AN INCORRECTLY RECEIVED HEADER_LENGTH FIELD. AN
                        NAK_PACKET IS GENERATED.
NOTES:                  FRAME SECONDARY INFORMATION PACKETS LAYER
                        EXECUTE CONTROL INCOMING PACKET LAYER


DATA ELEMENT NAME:      IDENTIFY
ALIASES:                NONE
VALUES AND MEANINGS:
                        USED TO PASS A SET OR CLEAR CONDITION FROM SNAK FLAG FILE
                        OR SACK FLAG FILE OR SREP FLAG FILE TO THE ASSOCIATED
                        GENERATE PROCESS.
NOTES:                  EXECUTE OUTGOING CONTROL PACKET LAYER


DATA ELEMENT NAME:      INCREMENT_R
ALIASES:                NONE
VALUES AND MEANINGS:
                        NEW VALUE OF R (HIGHEST SEQUENTIAL DATA MESSAGE RECEIVED)
                        USED TO UPDATE THE R_FILE AND TRIGGER THE SENDING OF AN
                        ACK_PACKET.
NOTES:                  EXECUTE INCOMING CONTROL PACKET LAYER
                        EXECUTE DATA PACKET LAYER


DATA ELEMENT NAME:      INITIALIZATION_COMPLETE
ALIASES:                NONE
VALUES AND MEANINGS:
                        RESPONSE TO INITIALIZE_LINK COMMAND
NOTES:                  EXECUTE INCOMING CONTROL PACKETS LAYER


DATA ELEMENT NAME:      INITIALIZATION_ON_OTHER_END
ALIASES:                NONE
```

VALUES AND MEANINGS:
THE OTHER END HAS BEEN INITIALIZED, HALT PROTOCOL.
NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER

DATA ELEMENT NAME: INITIALIZE_LINK
ALIASES: NONE
VALUES AND MEANINGS:
INITIALIZE THE PROTOCOL AND START THE DATA LINK
NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER

DATA FLOW NAME: INITIAL_PACKET
ALIASES: NONE
COMPOSITION:

```
                                    __          __
INITIAL_PACKET = 2{SYNC}8 + | SOH_PACKET |
                            | DLE_PACKET |
                            | ENQ_PACKET |
                            --          --
```

NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA ELEMENT NAME: INVALID_BLKCK1
ALIASES: NONE
VALUES AND MEANINGS:
FLAGS AN INCORRECTLY RECEIVED SOH/DLE HEADER. AN NAK_PACKET IS RETURNED TO THE TRANSMITTER.
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_BLKCK2
ALIASES: NONE
VALUES AND MEANINGS:
FLAGS AN INCORRECTLY RECEIVED DATA FIELD. AN NAK_PACKET IS GENERATED.
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_BLKCK3
ALIASES: NONE
VALUES AND MEANINGS:
FLAGS AN INCORRECTLY RECEIVED ENQ_HEADER. AN NAK_PACKET IS GENERATED.
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_DATA_LENGTH
ALIASES: NONE
VALUES AND MEANINGS:
FLAGS AN INCORRECTLY RECEIVED DATA LENGTH FIELD. AN NAK_PACKET IS GENERATED.
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE DATA PACKET LAYER

DATA ELEMENT NAME: INVALID_DLE_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
FLAGS A NON-SELECTED TRANSMISSION OWNERSHIP CONTROL BIT WHEN IN THE MAINTENANCE MODE
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_PACKETS
ALIASES: NONE
VALUES AND MEANINGS:
FLAG USED TO TRANSMIT AN NAK_PACKET OBTAINED FROM RECEPTION OF AN:

1. INVALID_BLKCK1
2. INVALID_BLKCK2
3. INVALID_BLKCK3
4. PACKET_NON_ABUT

NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_STACK_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG USED TO TRANSMITTER TO SEND OUT A NAK_PACKET SINCE THE SELECT BIT WAS NOT SET IN THE RECEIVED_STACK_PACKET RESULTING IN THE RECEPTION OF A BAD STACK_PACKET.
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: INVALID_STRT_PACKET
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG TO THE TRANSMITTER TO SEND OUT A NAK_PACKET SINCE THE SELECT BIT WAS NOT SET IN THE RECEIVED_STRT_PACKET RESULTING IN THE RECEPTION OF A BAD STRT_PACKET.

NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA FLOW NAME: LINK_AND_MODEM_CONTROL
ALIASES: NONE
COMPOSITION:

LINK_AND_MODEM_CONTROL = [DISABLE_LINK | ENABLE_LINK]

NOTES: START REPLY TIMER LAYER

DATA FLOW NAME: LINK_DATA_PACKET
ALIASES: NONE
COMPOSITION:

LINK_DATA_PACKET =

[TRANSPORT PROTOCOL + BLKCK2 + ADDR + BLKCK1 + COUNT + SOH + FLAGS | TRANSPORT PROTOCOL + BLKCK2 + ADDR + BLKCK1 + COUNT + SOH]

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: *LINK_PACKETS*
ALIASES: ABUTTED_PACKET
SYNC_PACKET
FOLLOWON_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: LOCALLY_GENERATED_CRC_REMAINDER
ALIASES: NONE
VALUES AND MEANINGS:
A SIXTEEN BIT SEQUENCE THAT IS THE REMAINDER AFTER DIVISION OF THE VALID_LENGTH_SOH/DLE_HEADER, VALID_LENGTH_DATA, OR VALID_LENGTH_ENQ_HEADER BY THE CRC POLYNOMIAL.
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: LONG_DATA_COUNT

ALIASES: SHORT_DATA_COUNT
COMPOSITION:

LONG_DATA_COUNT = TRANSPORT PROTOCOL + BLKCK2 + ADDR + SOH + COUNT

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: MODEM_FAILURE
ALIASES: NONE
VALUES AND MEANINGS:

A NOTIFICATION FROM THE DRIVER THAT THE MODEM/LINK HAS FAILED.

NOTES: START REPLY TIMER LAYER

DATA FLOW NAME: MOP_PACKETS
ALIASES: NONE
COMPOSITION:

MOP_PACKETS = | PARAMETER_LOAD_WITH_TRANSFER_ADDRESS |
| MEMORY_LOAD_WITH_TRANSFER_ADDRESS |
| REQUEST_MEMORY_DUMP |
| ENTER_MOP_MODE |
| REQUEST_PROGRAM |
| REQUEST_MEMORY_LOAD |
| MOP_MODE_RUNNING |
| MEMORY_DUMP_DATA |
| LOOP_BACK_TEST |
| MEMORY_LOAD_WITHOUT_TRANSFER_ADDRESS |

NOTES: EXECUTE MAINTENANCE PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: NAK_PACKET
ALIASES: RECEIVED_NAK_PACKET
COMPOSITION:

NAK_PACKET = ENQ + NAKTYPE + REASON + FLAGS + RESP + FILL + ADDR + BLKCK3

NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKET LAYER

DATA ELEMENT NAME: NAK_TRANSMIT_FLAG
ALIASES: SET_STARTUP
RESYNC
VALUES AND MEANINGS:

A FLAG USED TO SET THE CORRESPONDING BIT IN THE STARTUP_

FLAG_FILE.

NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: NAKTYPE
ALIASES: NONE
VALUES AND MEANINGS:

NAK PACKET TYPE = 00000010

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: N=EVEN
ALIASES: NONE
VALUES AND MEANINGS:

WHEN N (COUNTER) IS AN EVER NUMBER AND THE TRANSMITTER RECEIVES A VALID_STRT_PACKET, THEN RETURN A STACK_PACKET.

NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER

DATA ELEMENT NAME: N=ODD
ALIASES: NONE
VALUES AND MEANINGS:

WHEN N (COUNTER) IS A ODD NUMBER AND THE TRANSMITTER RECEIVES A VALID_STRT_PACKET, THEN RETURN A STRT_PACKET.

NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER

DATA FLOW NAME: NO_HEADER_DATA_PACKET
ALIASES: NONE
COMPOSITION:

NO_HEADER_DATA_PACKET = TRANSPORT PACKET + BLKCK2 + COUNT

NOTES: FRAMING PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: NOSYNC_PRIMARY_TO_SECONDARY_DATA_PACKET
ALIASES: COUNTED_LINK_DATA_PACKET
< 256_LINK_PACKET
RETRANSMITTED_PACKETS
COMPOSITION:

SEE ALIASES

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: NUM

ALIASES: NONE
VALUES AND MEANINGS:

USED TO DENOTE THE NUMBER OF THIS DATA PACKET. WHEN USED WITH REP_PACKET IT IS THE NUMBER OF THE LAST SEQUENTIALLY NUMBERED DATA PACKETS SENT. THIS IS COMPARED AGAINST THE NUMBER OF THE LAST SEQUENTIAL PACKET RECEIVED AND RESULTS IN EITHER AN ACK BEING RETURNED IF THEY AGREE OR A NAK IF THEY DO NOT.

NOTES: ALL LINK LAYERS
START REPLY TIMER LAYER

DATA ELEMENT NAME: PACKET_NON_ABUT
ALIASES: NONE
VALUES AND MEANINGS:

FLAG INDICATING THAT THE LAST TWO MESSAGES RECEIVED DID NOT ABUT AND THE TRANSMITTER SHOULD BE NOTIFIED TO RETRANSMIT.

NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER
EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: PERSISTENT_ERROR
ALIASES: NONE
VALUES AND MEANINGS:

AN ERROR HAS OCCURRED FROM WHICH RECOVERY IS NOT POSSIBLE.
THIS ERROR IS: 1. 7 CONSECTATIVE RESPONSES
2. TIMER EXPIRATIONS

NOTES: START REPLY TIMER LAYER

DATA FLOW NAME: PAD/STRT_PACKET
ALIASES: NONE
COMPOSITION:

PAD/STRT_PACKET = [11111111 + STRT_PACKET]
[STRT_PACKET]

NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: PHYSICAL_TO_LINK_PACKETS
ALIASES: NONE
COMPOSITION:

```
                    PHYSICAL_TO_LINK_PACKETS = (2{SYNC}8) + | SOH_PACKET |
                                                           | DLE_PACKET |
                                                           | ENQ_PACKET |
                                                           --           --
NOTES:              DECODE AND SYNC INCOMING BIT STREAM LAYER


DATA FLOW NAME:     PIGGYBACKED_LINK_DATA_PACKET
ALIASES:            NONE
COMPOSITION:
                    PIGGYBACKED_LINK_DATA_PACKET = LINK_DATA_PACKET + RESP
NOTES:              FRAME PRIMARY INFORMATION PACKETS LAYER


DATA FLOW NAME:     PRIMARY_INCOMING_BIT_STREAM_FROM_DRIVER
ALIASES:            PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER
                    SECONDARY_OUTGOING_BIT_STREAM_TO_DRIVER
                    SECONDARY_INCOMING_BIT_STREAM_FROM_DRIVER
COMPOSITION:
                    PRIMARY_INCOMING_BIT_STREAM_FROM_DRIVER = (11111111) +
                                                              __           __
                                          (2{SYNC}8) + | SOH_PACKET |
                                                       | DLE_PACKET |
                                                       | ENQ_PACKET |
                                                       --           --
NOTES:              DECODE AND SYNC INCOMING BIT STREAM LAYER
                    OVERVIEW LAYER
                    EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER


DATA FLOW NAME:     PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER
ALIASES:            PRIMARY_INCOMING_BIT_STREAM_FROM_DRIVER
                    SECONDARY_INCOMING_BIT_STREAM_FROM_DRIVER
                    SECONDARY_OUTGOING_BIT_STREAM_TO_DRIVER
COMPOSITION:
                    SEE ALIASES
NOTES:              OVERVIEW LAYER
                    EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
                    START REPLY TIMER LAYER


DATA FLOW NAME:     PRIMARY_TO_SECONDARY_DATA_PACKET
ALIASES:            NONE
COMPOSITION:
                    PRIMARY_TO_SECONDARY_DATA_PACKET = (2{SYNC}8) + LINK_
                                  DATA_PACKET + RESP + NUM
NOTES:              FRAME PRIMARY INFORMATION PACKETS LAYER
```

START REPLY TIMER LAYER
EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: PRIMARY_TO_SECONDARY_CONTROL_PACKET
ALIASES: NONE
COMPOSITION:

PRIMARY_TO_SECONDARY_CONTROL_PACKET = [PAD/STRT_PACKET | NAK_PACKET | STACK_PACKET | ACK_PACKET | REP_PACKET]

NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: PRIMARY_TO_SECONDARY_MAINTENANCE_PACKET
ALIASES: NONE
COMPOSITION:

PRIMARY_TO_SECONDARY_MAINTENANCE_PACKET =

[TRANSMIT_DLE_PACKET | MOP_PACKETS]

NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA ELEMENT NAME: QSYNC
ALIASES: NONE
VALUES AND MEANINGS:

QUICK SYNC FLAG, USED TO NOTIFY THE RECEIVER THAT THE NEXT PACKET WILL NOT ABUT THIS PACKET, AND RESYNCHRONIZATION SHOULD FOLLOW THIS PACKET.

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: QSYNC_DATA_PACKET
ALIASES: NONE
COMPOSITION:

QSYNC_DATA_PACKET = TRANSPORT PROTOCOL + BLKCK2 + ADDR + COUNT + SOH + FLAGS

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: QSYNC_ENQ/DLE_PACKET
ALIASES: ENQ/DLE_SYNC_PACKET

COMPOSITION:
SEE ALIASES
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: RCVR
ALIASES: NONE
VALUES AND MEANINGS:
CONTROL PACKET RECEIVER FIELD. USED TO PASS INFORMATION FROM THE DATA PACKET RECEIVER TO THE DATA PACKET SENDER.
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: REASON
ALIASES: NONE
VALUES AND MEANINGS:
NAK ERROR REASONS
1. ERROR DUE TO TRANSMISSION MEDIUM
000001 = HEADER BLOCK CHECK ERROR
000010 = DATA FIELD BLOCK CHECK ERROR
000011 = REP RESPONSE
2. ERROR DUE TO COMPUTERS/INTERFACE
001000 = BUFFER TEMPORARILY UNAVAILABLE
001001 = RECEIVE OVERRUN
010000 = PACKET TOO LONG
010001 = PACKET HEADER FORMAT ERROR
NOTES ALL LINK LAYERS

DATA FLOW NAME: RECEIVED_ACK_PACKET
ALIASES: ACK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE DATA PACKET LAYER
EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_NAK_PACKET
ALIASES: NAK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE DATA PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER
EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: RECEIVED_PRIMARY_NODE_TRANSPORT_PACKET
ALIASES: RECEIVED_SECONDARY_NODE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA FLOW NAME: RECEIVED_REP_PACKET
ALIASES: REP_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA FLOW NAME: RECEIVED_SECONDARY_NODE_TRANSPORT_PACKET
ALIASES: RECEIVED_PRIMARY_NODE_TRANSPORT_PACKET
COMPOSITION:
RECEIVED_SECONDARY_NODE_TRANSPORT_PACKET = TRANSPORT_TO_
LINK_DATA_PACKET
NOTES: OVERVIEW LAYER

DATA FLOW NAME: RECEIVED_STACK_PACKET
ALIASES: STACK_PACKET
VALID_STACK_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
START REPLY TIMER LAYER

DATA FLOW NAME: RECEIVED_STRT_PACKET
ALIASES: STRT_PACKET
VALID_STRT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
START REPLY TIMER LAYER

DATA ELEMENT NAME: RECEIVE_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
TRANSPORT PROTOCOL SUPPLIES A POOL OF BUFFERS TO THE LINK

PROTOCOL.

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: REP_PACKET
ALIASES: RECEIVED_REP_PACKET
COMPOSITION:

REP_PACKET = ENQ + REPTYPE + REPSUB + FLAGS + FILL +
NUM + ADDR + BLKCK3

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER

DATA ELEMENT NAME: REPSUB
ALIASES: NONE
VALUES AND MEANINGS:

REP SUBTYPE = 00000000

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: REPTYPE
ALIASES: NONE
VALUES AND MEANINGS:

REP PACKET TYPE = 00000011

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: RESET_SIGNAL
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG TO THE TIMER TO RESET SINCE A POSITIVE ACKNOWLEDGE
HAS BEEN RECEIVED.

NOTES: START REPLY TIMER LAYER

DATA ELEMENT NAME: RESET_STARTUP
ALIASES: NONE
VALUES AND MEANINGS:

A FLAG TO THE STARTUP_FLAG_FILE TO CLEAR THIS FILE.

NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA ELEMENT NAME: RESP
ALIASES: NONE
VALUES AND MEANINGS:

USED TO ACKNOWLEDGE CORRECTLY RECEIVED PACKETS (THE PIGGYBACKED ACK). THE NUMBER REPRESENTS THE LAST CONSECUTIVE CORRECTLY RECEIVED PACKET RECEIVED FROM THE ADDRESSED NODE BY THE NODE TRANSMITTING THIS PACKET. IT IMPLYS THAT ALL UNACKNOWLEDGED PACKETS BETWEEN THE ONE ACKNOWLEDGED IN THE LAST RESP FIELD RECEIVED AND THE ONE ACKNOWLEDGED BY THIS RESP FIELD (MOD 256), HAVE BEEN RECEIVED CORRECTLY. WHEN USED IN A NAK_PACKET, USUALLY IMPLYS SOME ERROR IN A PACKET WITH NUMBER RESP + 1 OR BEYOND.

NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: RESYNC
ALIASES: NAK_TRANSMIT_FLAG
SET_STARTUP
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA FLOW NAME: RETRANSMITTED_PACKETS
ALIASES: COUNTED_LINK_DATA_PACKET
< 256_LINK_PACKET
NOSYNC_PRIMARY_TO_SECONDARY_DATA_PACKET
COMPOSITION:
SEE ALIASES
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: SECONDARY_INCOMING_BIT_STREAM_FROM_DRIVER
ALIASES: PRIMARY_INCOMING_BIT_STREAM_FROM_DRIVER
PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER
SECONDARY_OUTGOING_BIT_STREAM_TO_DRIVER
COMPOSITION:
SEE ALIASES:
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SECONDARY_OUTGOING_BIT_STREAM_TO_DRIVER
ALIASES: PRIMARY_INCOMING_BIT_STREAM_FROM_DRIVER
PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER
SECONDARY_INCOMING_BIT_STREAM_FROM_DRIVER
COMPOSITION:
SEE ALIASES
NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: SET
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG TO SET COUNTERS, SNAK FLAG FILE OR SACK FLAG FILE.
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: SET_STARTUP
ALIASES: NAK_TRANSMIT_FLAG
RESYNC
VALUES AND MEANINGS:
SEE ALIASES
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA FLOW NAME: SHORT_DATA_COUNT
ALIASES: LONG_DATA_COUNT
COMPOSITION:
SEE ALIASES
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: SNDR
ALIASES: NONE
VALUES AND MEANINGS:
CONTROL PACKAT SENDER FIELD. USED TO PASS INFORMATION
FROM THE DATA PACKET SENDER TO THE DATA PACKET RECEIVER.
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: SOH
ALIASES: NONE
VALUES AND MEANINGS:
THE NUMBERED DATA MESSAGE IDENTIFIER = 10000001
NOTES: ALL LINK LAYERS

DATA FLOW NAME: SOH_PACKET
ALIASES: VALID_SOH_PACKET
COMPOSITION:
SOH_PACKET = SOH + COUNT + FLAGS + RESP + NUM + ADDR +
BLKCK1 + DATA + BLKCK2
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER
FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: STACK_PACKET
ALIASES: VALID_STACK_PACKET
RECEIVED_STACK_PACKET
COMPOSITION:
STACK_PACKET = ENQ + STCKTYPE + STCKSUB + FLAGS + FILL + FILL + ADDR + BLKCK3
NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: START
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG TO THE LOAD AND DELETE MEMORY PROCESS TO TRIGGER THE SENDING OF RETRANSMITTED_PACKETS.
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: STARTUP
ALIASES: NONE
VALUES AND MEANINGS:
A FLAG TO THE "CHECK FOR STARTUP" PROCESS TO SET OR CLEAR THE STARTUP BIT.
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA ELEMENT NAME: STCKSUB
ALIASES: NONE
VALUES AND MEANINGS:
STACK SUBTYPE = 00000000
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: STCKTYPE
ALIASES: NONE
VALUES AND MEANINGS:
STACK PACKET TYPE = 00000111
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: STOP_LINK
ALIASES: NONE
VALUES AND MEANINGS:

HALT THE PROTOCOL
NOTES: START REPLY TIMER LAYER

DATA FLOW NAME: STRT_PACKET
ALIASES: VALID_STRT_PACKET
RECEIVED_STRT_PACKET
COMPOSITION:
STRT_PACKET = ENQ + STRTTYPE + STRTSUB + FLAGS + FILL +
FILL + ADDR + BLKCK3
NOTES: EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: STRTSUB
ALIASES: NONE
VALUES AND MEANINGS:
STRT SUBTYPE = 00000000
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: STRTTYPE
ALIASES: NONE
VALUES AND MEANINGS:
STRT PACKET TYPE = 00000110
NOTES: ALL LINK LAYERS

DATA FLOW NAME: SUBTYPE
ALIASES: NONE
COMPOSITION:
SUBTYPE = [(ACKSUB) + (REPSUB) + (STRTSUB) + (STCKSUB)]
NOTES: ALL LINK LAYERS

DATA ELEMENT NAME: SYNC
ALIASES: NONE
VALUES AND MEANINGS:
SYNCHRONIZATION BYTE, THE OCTET 10010110
NOTES: OVERVIEW LAYER

DATA FLOW NAME: SYNC_PACKET
ALIASES: FOLLOWON_PACKET
ABUTTED_PACKET

LINK_PACKETS

COMPOSITION:

SEE ALIASES

NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

DATA FLOW NAME: TIMEOUT
ALIASES: NONE
COMPOSITION:

TIMEOUT = INCREMENT ERROR COUNTER + [TIMEOUT_DATA | TIMEOUT_STRT | TIMEOUT_STACK]

NOTES: START REPLY TIMER LAYER

DATA ELEMENT NAME: TIMEOUT_DATA
ALIASES: NONE
VALUES AND MEANINGS:

AN ACKNOWLEDGEMENT TO DATA PACKET IS HELD UP AND RESULTS IN THE REPLY TIMER EXPIRATION CAUSING A TIMEOUT SIGNAL TO TRIGGER THE SENDING OF A REP_PACKET.

NOTES: START REPLY TIMER LAYER.

DATA ELEMENT NAME: TIMEOUT_STACK
ALIASES: NONE
VALUES AND MEANINGS:

AN ACKNOWLEDGEMENT TO A STACK_PACKET IS HELD UP AND RESULTS IN THE REPLY TIMER EXPIRATION CAUSING A TIMEOUT SIGNAL TO TRIGGER THE SENDING OF A STACK_PACKET.

NOTES: START REPLY TIMER LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER

DATA ELEMENT NAME: TIMEOUT_STRT
ALIASES: NONE
VALUES AND MEANINGS:

AN ACKNOWLEDGEMENT TO A START PACKET IS HELD UP AND RESULTS IN THE REPLY TIMER EXPIRATION CAUSING A TIMEOUT SIGNAL TO TRIGGER THE RESENDING OF A STRT_PACKET.

NOTES: START REPLY TIMER LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER

DATA ELEMENT NAME: TRANSMIT_COMPLETE

ALIASES: NONE
VALUES AND MEANINGS:
A NOTIFICATION TO THE LINK PROTOCOL ONCE THE DRIVER HAS COMPLETED A PREVIOUS TRANSMIT_A_BLOCK COMMAND
NOTES: START REPLY TIMER LAYER

DATA FLOW NAME: TRANSMIT_DLE_PACKET
ALIASES: DLE_PACKET
VALID_DLE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE MAINTENANCE PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA ELEMENT NAME: TRANSIENT_ERROR_THRESHOLD_COUNTER_OVERFLOW
ALIASES: NONE
VALUES AND MEANINGS:
ERROR_THRESHOLD COUNTER OVERFLOW DUE TO
1. NAKS RECEIVED
2. ERRORS THAT CAUSE NAKS TO BE SENT
THIS COMMAND AUTOMATICLY CLEARS THE TRESHOLD_COUNTER WHEN THE NUMBER OF ERRORS RECORDED EQUALS 7.
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA ELEMENT NAME: TRANSMIT_A_ BLOCK
ALIASES: NONE
VALUES AND MEANINGS:
PASS A BLOCK TO THE DRIVER FOR TRANSMISSION.
NOTES: START REPLY TIMER LAYER

DATA ELEMENT NAME: TRANSMIT_PACKET
ALIASES: NONE
VALUES AND MEANINGS:
GIVE A PACKET TO LINK PROTOCOL FOR TRANSMISSION.
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: TRANSMITTED_PRIMARY_NODE_TRANSPORT_PACKET
ALIASES: TRANSMITTED_SECONDARY_NODE_TRANSPORT_PACKET
COMPOSITION:
SEE ALIASES
NOTES: EXECUTE DATA PACKET LAYER

OVERVIEW LAYER
EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

DATA FLOW NAME: TRANSMITTED_SECONDARY_NODE_TRANSPORT_PACKET
ALIASES: TRANSMITTED_PRIMARY_NODE_TRANSPORT_PACKET
COMPOSITION:

TRANSMITTED_SECONDARY_NODE_TRANSPORT_PACKET = VALID_SOH_PACKET

NOTES: OVERVIEW LAYER

DATA ELEMENT NAME: TRANSPORT_TO_LINK_DATA_PACKET
ALIASES: NONE
VALUES AND MEANINGS:

ANY TRANSPORT PACKET

NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

DATA FLOW NAME: TYPE
ALIASES: NONE
COMPOSITION:

TYPE = | ATCKTYPE |
| STRTTYPE |
| REPTYPE |
| NAK_TYPE |
| ACK_TYPE |

NOTES: ALL LINK LAYERS

DATA FLOW NAME: VALID_CRC_ENQ_PACKET
ALIASES: ENQ_PACKET
VALID_ENQ_PACKET
VALID_LENGTH_ENQ_HEADER
COMPOSITION:

SEE ALIASES
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_CRC_SOH/DLE_PACKET
ALIASES: VALID_LENGTH_SOH/DLE_HEADER
VALID_SOH/DLE_HEADER
VALID_LENGTH_DATA
COMPOSITION:

VALID_CRC_CRC/DLE_PACKET =

| SOH + COUNT + FLAGS + RESP + NUM + ADDR + BLKCK1 + DATA + BLKCK2 |
| DLE + COUNT + FLAGS + FILL + FILL + ADDR + BLKCK1 + DATA + BLKCK2 |

NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_DLE_PACKET
ALIASES: DLE_PACKET
COMPOSITION:
SEE ALIASES
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_ENQ_PACKET
ALIASES: ENQ_PACKET
VALID_CRC_ENQ_PACKET
VALID-LENGTH_ENQ_HEADER
COMPOSITION:
SEE ALIASES
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE INCOMING CONTROL PACKET LAYER

DATA FLOW NAME: VALID_LENGTH_DATA
ALIASES: VALID_CRC_SOH/DLE_PACKET
VALID_LENGTH_SOH/DLE_PACKET
VALID_SOH/DLE_HEADER
COMPOSITION:
SEE ALIASES
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_LENGTH_ENQ_HEADER
ALIASES: ENQ_PACKET
VALID_ENQ_PACKET
VALID_CRC_ENQ_PACKET
COMPOSITION:
SEE ALIASES
NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_LENGTH_SOH/DLE_HEADER
ALIASES: VALID_CRC_SOH/DLE_HEADER

VALID_LENGTH_DATA

COMPOSITION:

SEE ALIASES

NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_SOH/DLE_HEADER

ALIASES: VALID_CRC_SOH/DLE_PACKET
VALID_LENGTH_DATA
VALID_LENGTH_SOH/DLE_HEADER

COMPOSITION: SEE ALIASES

NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER

DATA FLOW NAME: VALID_SOH_PACKET

ALIASES: SOH_PACKET

COMPOSITION:

SEE ALIASES

NOTES: FRAME SECONDARY INFORMATION PACKETS LAYER
EXECUTE DATA PACKET LAYER

DATA FLOW NAME: VALID_STACK_PACKET

ALIASES: STACK_PACKET
RECEIVED_STACK_PACKET

COMPOSITION:

SEE ALIASES

NOTES: EXECUTE INCOMING CONTROL PACKET LAYER

DATA FLOW NAME: VALID_STRT_PACKET

ALIASES: STRT_PACKET
RECEIVED_STRT_PACKET

COMPOSITION:

SEE ALIASES

NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER

DATA FLOW NAME: < 256_LINK_PACKET

ALIASES: COUNTED_LINK_DATA_PACKET
RETRANSMITTED_PACKETS
NOSYNC_PRIMARY_TO_SECONDARY_DATA_PACKET

COMPOSITION:

SEE ALIASES

NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

FILE DEFINITIONS (L)

FILE OR DATABASE NAME: A_FILE
ALIASES: NONE
COMPOSITION:
THE NUMBER OF THE HIGHEST SEQUENTIAL DATA MESSAGE THAT HAS BEEN ACKNOWLEDGED TO THIS STATION. RECEIVED IN THE RESP FIELD OF DATA_PACKETS, ACK_PACKETS, AND NAK_PACKETS.
ORGANIZATION: SINGLE OCTET
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE DATA PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER
START REPLY TIMER LAYER

FILE OR DATABASE NAME: HOLD_TABLE
ALIASES: NONE
COMPOSITION:
HOLD_TABLE = LINK_DATA_PACKET
ORGANIZATION: VARIABLE-LENGTH OCTET
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

FILE OR DATABASE NAME: LINK_TO_PHYSICAL_COMMAND_TABLE
ALIASES: NONE
COMPOSITION:
LINK_TO_PHYSICAL-COMMAND_TABLE =
(LINK_AND_MODEM_CONTROL) +
(TRANSMIT_A_BLOCK)
ORGANIZATION: 20-BIT TABLE
NOTES: START REPLY TIMER LAYER

FILE OR DATABASE NAME: LINK_TO_TRANSPORT_COMMAND_TABLE
ALIASES: NONE
COMPOSITION:
LINK_TO_TRANSPORT_COMMAND_TABLE = (PERSISTENT_ERROR) +
(TRANSIENT_ERROR_THRESHOLD_COUNTER_OVERFLOW) +
(INITIALIZATION_ON_OTHER_END) +
(INITIALIZATION_COMPLETE)
ORGANIZATION: 4-BIT TABLE
NOTES: EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER
START REPLY TIMER LAYER

FILE OR DATABASE NAME: MEMORY

ALIASES: NONE
COMPOSITION:

STANDARD COMPUTER MEMORY. ALL DATA PACKETS SENT ARE RECORDED IN MEMORY AND WHEN A VALID ACKNOWLEDGE IS RECEIVED THEY ARE DELETED.

ORGANIZATION: VARIABLE LENGTH 8-BIT-BYTE WORDS
NOTES: FRAME PRIMARY INFORMATION PACKETS LAYER

FILE OR DATABASE NAME: PHYSICAL_TO_LINK_COMMAND_TABLE
ALIASES: NONE
COMPOSITION:

PHYSICAL_TO_LINK_COMMAND_TABLE = (TRANSMIT_COMPLETE) + (MODEM_FAILURE)

ORGANIZATION: 2-BIT TABLE
NOTES: START REPLY TIMER LAYER

FILE OR DATABASE NAME: R_FILE
ALIASES: NONE
COMPOSITION:

THE NUMBER OF THE HIGHEST SEQUENTIAL DATA PACKET RECEIVED AT THIS STATION. SENT IN THE RESP FIELD OF DATA_PACKETS, ACK_PACKETS, AND NAK_PACKETS AS ACKNOWLEDGEMENT TO THE OTHER STATION.

ORGANIZATION: SINGLE OCTET
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE DATA PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

FILE OR DATABASE NAME: SACK
ALIASES: NONE
COMPOSITION:

THIS FLAG IS SET WHEN EITHER THE R-FILE IS INCREMENTED MEANING A NEW SEQUENTIAL DATA PACKET HAS BEEN RECEIVED WHICH REQUIRES AN ACK REPLY. THE SACK FLAG IS CLEARED WHEN SENDING EITHER A DATA PACKET WITH THE LATEST RESP FIELD INFORMATION, OR AN ACK WITH THE LATEST RESP FIELD INFORMATION.

ORGANIZATION: SINGLE-BIT VARIABLE
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE INCOMING CONTROL PACKET LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

FILE OR DATABASE NAME: SNAK
ALIASES: NONE
COMPOSITION:
THIS FLAG IS SET WHEN A RECEIVE ERROR OCCURS THAT REQUIRES A NAK REPLY. IF IS CLEARED WHEN A NAK_PACKET IS SENT WITH LATEST RESP INFORMATION. ADDITIONALLY A NAK REASON VARIALBE IS SET IN THE FILE.

ORGANIZATION: OC

FILE OR DATABASE NAME: SREP
ALIASES: NONE
COMPOSITION:
THIS FLAG IS SET WHEN A REPLY TIMER EXPIRES IN THE RUNNING STATE AND A REP SHOULD BE SENT. IT IS CLEARED WHEN AN REP_PACKET IS SENT.

ORGANIZATION: SINGLE-BIT VARIABLE
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER
START REPLY TIMER LAYER

FILE OR DATABASE NAME: STARTUP_FLAG_FILE
ALIASES: NONE
COMPOSITION:
A FLAG THAT IS SET WHEN
1. STARTUP-PAD SEQUENCE IDENTIFIED
2. TO LOCATE THE NEXT PACKET AFTER A NAK_ PACKET IS SENT.
3. TO LOCATE THE NEXT PACKET AFTER A QSYNC_ PACKET IS SENT.

THE FLAG IS CLEARED ON THE RECEPTION OF A NORMAL (NO_PAD SEQUENCE) PACKET.

ORGANZATION: SINGLE-BIT VARIABLE
NOTES: DECODE AND SYNC INCOMING BIT STREAM LAYER

FILE OR DATABASE NAME: T_FILE
ALIASES: NONE
COMPOSITION:
THE NUMBER OF THE NEXT DATA_PACKET TO BE TRANSMITTED. WHEN SENDING NEW DATA_PACKETS, T WILL HAVE THE VALUE V + 1. WHEN RETRANSMITTING, T WILL BE SET BACK TO A + 1 AND WILL ADVANCE TO V + 1.

ORGANZATION: SINGLE OCTET
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER

FILE OR DATABASE NAME: TRANSPORT_TO_LINK_COMMAND_TABLE
ALIASES: NONE
COMPOSITION:

```
TRANSPORT_TO_LINK_COMMAND_TABLE = (STOP_LINK) +
            (INITIALIZE_LINK) + (RECEIVE_PACKET) +
            (TRANSMIT_PACKET) +
            (ENTER_MAINTENACE_MODE)
```

ORGAIZATION: 5-BIT TABLE
NOTES: START REPLY TIMER LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER
EXECUTE MAINTENANCE PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

FILE OR DATABASE NAME: V_FILE
ALIASES: NONE
COMPOSITION:

THE NUMBER OF THE HIGHEST SEQUENTIAL DATA_PACKET TRANSMITTED BY THIS STATION. SENT IN THE NUM FIELD OF REP_PACKETS. V IS THE NUMBER ASSIGNED TO THE LAST USER TRANSMIT REQUEST WHICH HAS BEEN TRANSMITTED (SENT IN THE NUM FIELD OF THAT DAT PACKET).

ORGANIZATION: SINGLE OCTET
NOTES: EXECUTE HDLC PROTOCOL AT PRIMARY NODE LAYER
EXECUTE OUTGOING CONTROL PACKET LAYER
FRAME PRIMARY INFORMATION PACKETS LAYER

PROCESS SPECIFICATIONS (L)

```
PROCESS NAME:          CHECK FOR AND REMOVE PAD SEQUENCE
PROCESS NUMBER:        1.1.1
PROCESS DESCRIPTION:
IF 111111111001011010010110 is encountered then
     Delete the 11111111
     Output a SET_STARTUP_FLAG
     Output next consecutive bytes
ELSE output the next consecutive bytes


PROCESS NAME:          CHECK FOR STARTUP
PROCESS NUMBER:        1.1.2
PROCESS DESCRIPTION:
IF STARTUP flag set then
     Output next packet as INITIAL_PACKET
     Reset STARTUP
ELSE output next packet as FOLLOWON_PACKET


PROCESS NAME:          LOCATE TWO CONSECUTIVE SYNC BYTES
PROCESS NUMBER:        1.1.3
PROCESS DESCRIPTION:
STRIP off all sync bytes
     Output remaining bytes as SYNC_PACKETS


PROCESS NAME:          DECODE PACKET HEADER
PROCESS NUMBER:        1.1.4
PROCESS DESCRIPTION:
IF first byte = 10000001 then
     Output ABUTTED_PACKET (SOH_PACKET)
ELSEIF first byte = 00000101 then
     Output ABUTTED_PACKET (ENQ_PACKET)
ELSEIF first byte = 10010000 then
     Output ABUTTED_PACKET (DLE_PACKET)
ELSE output PACKET_NON_ABUT


PROCESS NAME:          CHECK FOR QSYNC
PROCESS NUMBER:        1.1.5
PROCESS DESCRIPTION:
IF 23rd bit set then
     Output RESYNC to set STARTUP_FLAG_FILE
     Output SOH, DLE, or ENQ_PACKET
ELSE output SOH, DLE, or ENQ_PACKET
```

```
PROCESS NAME:          CHECK HEADER LENGTH
PROCESS NUMBER:        1.2.1
PROCESS DESCRIPTION:
IF SOH_PACKET or DLE_PACKET headers are less than 48 bits long then
     Output HEADER_FORMAT_ERROR
ELSE output VALID_LENGTH_SOH/DLE_HEADER


PROCESS NAME:          CHECK BLKCK1
PROCESS NUMBER:        1.2.2
PROCESS DESCRIPTION:
IF the BLKCK1 and the LOCALLY_GENERATED_CRC_REMAINDER are equal then
     Output VALID_LENGTH_SOH/DLE_HEADER as a VALID_SOH/DLE_HEADER
ELSE output as a INVALID_BLKCK1


PROCESS NAME:          CHECK DATA LENGTH (COUNT)
PROCESS NUMBER:        1.2.3
PROCESS DESCRIPTION:
IF SOH_PACKET or DLE_PACKET data fields are less than the value recorded in
     the COUNT field then
     Output VALID_SOH/DLE_HEADER as an INVALID_DATA_LENGTH
ELSE output as a INVALID_LENGTH_DATA


PROCESS NAME:          CHECK BLKCK2
PROCESS NUMBER:        1.2.4
PROCESS DESCRIPTION:
IF the BLKCK2 and the LOCALLY_GENERATED_CRC_REMAINDER are equal then
     Output VALID_LENGTH_DATA as a VALID_CRC_SOH/DLE_PACKET
ELSE output as a INVALID_BLKCK2


PROCESS NAME:          CHECK ENQ HEADER LENGTH
PROCESS NUMBER:        1.2.5
PROCESS DESCRIPTION:
IF ENQ_PACKET header is less than 48 bits long then
     Output HEADER_FORMAT_ERROR
ELSE output as VALID_LENGTH_ENQ_HEADER


PROCESS NAME:          CHECK BLKCK3
PROCESS NUMBER:        1.2.6
PROCESS DESCRIPTION:
```

```
IF the BLKCK3  and the LOCALLY_GENERATED_CRC_REMAINDER are equal then
     Output the VALID_LENGTH_ENQ_HEADER as VALID_CRC_ENQ_PACKET
ELSE output INVALID_BLKCK3
```

```
PROCESS NAME:          GENERATE CRC REMAINDER
PROCESS NUMBER:        1.2.7
PROCESS DESCRIPTION:
APPEND 16 zeros after the header or data protion of the packet (VALID_LENGTH_
     SOH/DLE_HEADER, VALID_LENGTH_DATA, VALID_LENGTH_ENQ_HEADER)
     TAKE the string of bits constructed and treat each bit as the coefficient
          of a term of a ploynomial with the LSB of the first byte being the
          coefficient of the highest order polynomial term. The highest order
          term is A * X ** 63 for a header block and A * X ** (8 * (count) +
          15) for a data block where A is the least significant bit of the
          first byte of the header or data. The lowest order term is 0 *
          X ** 0 for both cases.
          DIVIDE the constructed polynomial by the CRC-16 polynomial X ** 16 +
          X ** 15 + X ** 2 + 1 modulo 2, obtaining a quotient that is
          discarded and the 16-bit remainder.
               TRANSMIT the remainder as the LOCALLY_GENERATED_CRC_REMAINDER
```

```
PROCESS NAME:          CHECK SELECT BIT
PROCESS NUMBER:        1.2.8
PROCESS DESCRIPTION:
SEPARATE VALID_CRC_SOH/DLE_PACKET into VALID_SOH_PACKET and VALID_DLE_PACKET
     IF VALID_DLE_PACKET select bit not set then
          Output INVALID_DLE_PACKET
ELSE output VALID_DLE_PACKET
     Output VALID_SOH_PACKET
     Output VALID_CRC_ENQ_PACKET as VALID_ENQ_PACKET
```

```
PROCESS NAME:          DECODE ENQ PACKET
PROCESS NUMBER:        1.3.1
PROCESS DESCRIPTION:
IF TYPE equal 00000001 then
     Output RECEIVED_ACK_PACKET
ELSEIF TYPE equal 00000010 then
     Output RECEIVED_NAK_PACKET
ELSEIF TYPE equal 00000011 then
     Output RECEIVED_REP_PACKET
ELSEIF TYPE equal 00000110 then
     Output RECEIVED_STRT_PACKET
ELSE output RECEIVED_STACK_PACKET
```

```
PROCESS NAME:          SET NAK TRANSMIT FLAG
PROCESS NUMBER:        1.3.2
PROCESS DESCRIPTION:
IF Input = INVALID_BLKCK1, INVALID_BLKCK2, INVALID_BLKCK3, or PACKET_NON_
     ABUT then
     Output a NAK_TRANSMIT_FLAG and
     Output an INVALID_PACKETS flag
ELSE Null



PROCESS NAME:          SET NEGATIVE ACKNOWLEDGE FLAG, RESET TIMER
PROCESS NUMBER:        1.3.3
PROCESS DESCRIPTION:
IF Input = HEADER_FORMAT_ERROR, INVALID_DATA_LENGTH, INVALID_DLE_PACKET,
     INVALID_STRT_PACKET, INVALID_STACK_PACKET, DISAGREE, or INVALID_PACKETS
     flag then
     Output a SET flag to SNAK_FILE and to COUNT_ERRORS_PROCESS
     Reset REPLY TIMER
ELSE Null



PROCESS NAME:          CHECK STRT SELECT BIT
PROCESS NUMBER:        1.3.4
PROCESS DESCRIPTION:
IF RECEIVED_STRT_PACKET select bit set then
     Output VALID_STRT_PACKET
ELSE output INVALID_STRT_PACKET



PROCESS NAME:          CHECK STACK SELECT BIT
PROCESS NUMBER:        1.3.5
PROCESS DESCRIPTION:
IF RECEIVED_STACK_PACKET select bit set then
     Output VALID_STACK_PACKET
ELSE output INVALID_STACK_PACKET



PROCESS NAME:          CHECK NUM FIELD TO R
PROCESS NUMBER:        1.3.6
PROCESS DESCRIPTION:
IF RECEIVED_REP_PACKET NUM field value equal to the current R_FILE value then
     Output AGREE
ELSE output DISAGREE
```

```
PROCESS NAME:          SET ACKNOWLEDGE FLAG, RESET TIMER
PROCESS NUMBER:        1.3.7
PROCESS DESCRIPTION:
IF Input =  AGREE, INCREMENT_R, or VALID_STACK_PACKET then
     Output a SET flag to SACK_FILE
ELSE Null


PROCESS NAME:          COUNT ERRORS (NAK)
PROCESS NUMBER:        1.3.8
PROCESS DESCRIPTION:
INCREMENT counter on SET input
     IF counter = 49 then
          Output TRANSIENT_ERROR_THRESHOLD_COUNTER_OVERFLOW
          Reset counter to 0
     ELSE Null


PROCESS NAME:          N=N+1
PROCESS NUMBER:        1.4.1
PROCESS DESCRIPTION:
OUTPUT N = ODD when
     Reception of INITIALIZE_LINK (N=1)
     Reception of VALID_STRT_PACKET (N=ODD)
OUTPUT N = EVEN when
     Reception of VALID_STRT_PACKET (N=EVEN)


PROCESS NAME:          GENERATE NEGATIVE ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.2
PROCESS DESCRIPTION:
IF SNAK_FILE set then
     Output NAK_PACKET of the following format: ENQ + NAKTYPE + REASON +
                                                FLAGS + RESP + FILL + ADDR +
                                                BLKCK3
ELSE Null


PROCESS NAME:          GENERATE START PACKET
PROCESS NUMBER:        1.4.3
PROCESS DESCRIPTION:
IF N = ODD or TIMEOUT_STRT is inputed then
     Output STRT_PACKET of the following format: ENQ + STRTTYPE + STRTSUB +
                                                 FLAGS + FILL + FILL + ADDR +
                                                 BLKCK3
ELSE Null
```

```
PROCESS NAME:          GENERATE START ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.4
PROCESS DESCRIPTION:
IF N = EVER or TIMEOUT_STACK is inputted then
     Output STACK_PACKET of the following format: ENQ + STCKTYPE + STCKSUB +
                                                  FLAGS + FILL + FILL + ADDR +
                                                  BLKCK3

     Once STACK_PACKET is transmitted then
          Output INITIALIZATION_ON_OTHER_END flag
ELSE Null



PROCESS NAME:          GENERATE ACKNOWLEDGE PACKET
PROCESS NUMBER:        1.4.5
PROCESS DESCRIPTION:
IF SACK_FILE is set then
     Output as ACK_PACKET of the following format: ENQ + ACKTYPE + ACKSUB +
                                                  FLAGS + RESP + FILL +
                                                  ADDR + BLKCK3
ELSE Null



PROCESS NAME:          GENERATE REP PACKET
PROCESS NUMBER:        1.4.6
PROCESS DESCRIPTION:
IF SREP_FILE set then
     Output an REP_PACKET of the following format: ENQ + REPTYPE + REPSUB +
                                                  FLAGS + FILL + NUM +
                                                  ADDR + BLKCK3
ELSE Null



PROCESS NAME:          CLEAR NEGATIVE ACKNOWLEDGE FLAG
PROCESS NUMBER:        1.4.7
PROCESS DESCRIPTION:
UPON transmission of a NAK_PACKET
     Output flag to clear the SNAK_FILE



PROCESS NAME:          IF INITIALIZATION THEN PAD PACKET
PROCESS NUMBER:        1.4.8
PROCESS DESCRIPTION:
IF Input is INITIALIZE_LINK then
    Add PAD SEQUENCE (11111111) to STRT_PACKET
```

```
ELSE pass STRT_PACKET as is


PROCESS NAME:          CLEAR ACKNOWLEDGE FLAG
PROCESS NUMBER:        1.4.9
PROCESS DESCRIPTION:
UPON transmission of a ACK_PACKET
     Output flag to clear the SNAK_FILE


PROCESS NAME:          CLEAR SREP FLAG
PROCESS NUMBER:        1.4.10
PROCESS DESCRIPTION:
UPON transmission of a REP_PACKET
     Output flag to clear the SREP_FILE


PROCESS NAME:          PROCESS NUM FIELD
PROCESS NUMBER:        1.5.1
PROCESS DESCRIPTION:
SET R = NUM field value
     Output R value as INCREMENT_R to R_FILE
     Output VALID_SOH_PACKET


PROCESS NAME:          PROCESS RESP FIELD
PROCESS NUMBER:        1.5.2
PROCESS DESCRIPTION:
IF RESP field value from a VALID_SOH_PACKET or RECEIVED_ACK_PACKET or INVALID_
     DATA_LENGTH or RECEIVED_NAK_PACKET is greater than A_FILE value then
     SET A = RESP field value
     Output VALID_SOH_PACKET as TRANSMITTED_PRIMARY_NODE_TRANSPORT_PACKET
     Output a value to A_FILE
ELSE output VALID_SOH_PACKET as TRANSMITTED_PRIMARY_NODE_TRANSPORT_PACKET


PROCESS NAME:          GENERATE MAINTENANCE MODE PACKETS
PROCESS NUMBER:        1.6.1
PROCESS DESCRIPTION:
UPON reception of a VALID_DLE_PACKET
     Start maintenance operation protocol which issues MOP_PACKETS


PROCESS NAME:          GENERATE MAINTENANCE COMMAND
PROCESS NUMBER:        1.6.2
```

```
PROCESS DESCRIPTION:
OUTPUT a TRANSMIT_DLE_PACKET on input of an ENTER_MAINTENANCE_MODE_COMMAND
     with the following format: DLE + COUNT + FLAGS + FILL + FILL + ADDR +
                                BLKCK1 + DATA + BLKCK2


PROCESS NAME:          ADD 4 SYNC BYTES TO PACKET IF QSYNC
PROCESS NUMBER:        1.7.1
PROCESS DESCRIPTION:
IF QSYNC set in last packet then
     Add 4 SYNC bytes to the following packets depending on which is the next
          packet to arrive: NAK_PACKETS, ACK_PACKET, REP_PACKET, MOP_PACKET,
          TRANSMIT_DLE_PACKET
     Reset QSYNC bit once the 4 SYNC bytes have been added to the appropriate
          packet
     Add 4 SYNC bytes to the following packets at all times:  STACK_PACKET,
          and STRT_PACKET
     Output ENQ/DLE_SYNC_PACKET
ELSE add 4 SYNC bytes to the following packets at all times:  STACK_PACKET,
     and STRT_PACKET
     Output ENQ/DLE_SYNC_PACKET


PROCESS NAME:          COUNT DATA FIELD ADD BLKCK2 AND COUNT
PROCESS NUMBER:        1.7.2
PROCESS DESCRIPTION:
IF TRANSMIT_PACKET and RECEIVE_PACKET present then
     Accept incoming TRANSPORT_TO_LINK_DATA_PACKET
     Determine the number of bits in the TRANSPORT_TO_LINK_DATA_PACKET
     Put this number in the count field and use it to
     Generate BLKCK2
     Output NOHEADER_DATA_PACKET
ELSE Null


PROCESS NAME:          ADD REMAINING HEADER FIELDS
PROCESS NUMBER:        1.7.3
PROCESS DESCRIPTION:
TO NOHEADER_DATA_PACKET add ADDR field, NUM field, FLAGS field, RESP field,
     and SOH field
     IF COUNT field is less than or equal to 4 SYNC bytes then
          Output SHORT_DATA_COUNT
     ELSE output LONG_DATA_COUNT


PROCESS NAME:          SET QSYNC BIT
PROCESS NUMBER:        1.7.4
```

```
PROCESS DESCRIPTION:
IF Input is PAD/STRT_PACKET or STACK_PACKET then
     Set select bit and
     Output QSYNC_ENQ/DLE_PACKET
     Output QSYNC
ELSEIF Input is SHORT_DATA_COUNT then
     Set select bit and
     Output QSYNC_DATA_PACKET
ELSE output QSYNC_ENQ/DLE_PACKET
```

```
PROCESS NAME:          COUNT HEADER FIELD AND ADD BLKCK1
PROCESS NUMBER:        1.7.5
PROCESS DESCRIPTION:
TO LONG_DATA_COUNT and QSYNC_DATA_PACKET add BLKCK1
     Output LINK_DATA_PACKET
```

```
PROCESS NAME:          PIGGYBACK-ACKNOWLEDGE RECEIVED, SET RESP
PROCESS NUMBER:        1.7.6
PROCESS DESCRIPTION:
ADD T_FILE value to LINK_DATA_PACKET
     Clear SACK_FILE
     Output PIGGYBACK_LINK_DATA_PACKET
```

```
PROCESS NAME:          CHECK MODULO 256
PROCESS NUMBER:        1.7.7
PROCESS DESCRIPTION:
IF -255 < (A + 1 - V) < 1 and HOLD_TABLE contains data packet entries then
     Read  off and delete first data packet entry from top of table.
     Output as <256_LINK_PACKET
ELSEIF 1 < (A + 1 - V) < 256  and HOLD_TABLE contains data packet entries then
     Read off and delete first data packet entry from top of table
     Output as <256_LINK_PACKET
ELSEIF -255 < (A + 1 - V) < 1 or 1 < (A + 1 - V) < 256 then
     Output PIGGYBACKED_LINKED_DATA_PACKET as <256_LINK_PACKET
ELSE output PIGGYBACKED_LINKED_DATA_PACKET to bottom of HOLD_TABLE
```

```
PROCESS NAME:          INCREMENT PACKET COUNT
PROCESS NUMBER:        1.7.8
PROCESS DESCRIPTION:
IF V < 256 then
     V = V + 1
     T = V + 1
     Assign NUM = V
```

```
     Output COUNTED_LINK_DATA_PACKET
     Update V_FILE and T_FILE
ELSE V = 1
     T = 2
     Assign NUM = V
     Output COUNTED_LINK_DATA_PACKET
     Update V_FILE and T_FILE
```

```
PROCESS NAME:          CHECK RETRANSMIT
PROCESS NUMBER:        1.7.10
PROCESS DESCRIPTION:
IF T - V = 1 then
     Output COUNTED_LINK_DATA_PACKET to NOSYNC_PRIMARY_TO_SECONDARY_DATA_
          PACKET
ELSE output START flag
```

```
PROCESS NAME:          ADD 4 SYNC BYTES TO DATA PACKET IF QSYNC OR INIT
PROCESS NUMBER:        1.7.11
PROCESS DESCRIPTION:
IF Input is a QSYNC flag or V_FILE value of 1 or the first packet of the
     RETRANSMIT_PACKETS then
     Add 4 SYNC bytes and
     Output as PRIMARY_TO_SECONDARY_DATA_PACKET
ELSE output as PRIMARY_TO_SECONDARY_DATA_PACKET
```

```
PROCESS NAME:          LOAD AND DELETE MEMORY
PROCESS NUMBER:        1.7.12
PROCESS DESCRIPTION:
IF COUNTED_LINK_DATA_PACKET NM field value is greater than the A_FILE value
     then
     Load COUNTED_LINK_DATA_PACKET to MEMORY
ELSEIF R_FILE value is same as a recorded COUNTED_LINK_DATA_PACKET then
     Delete all COUNTED_LINK_DATA_PACKETS in MEMORY with NUM file values
          less than or equal to the new R_FILE value
ELSE receive a START flag then
     Output all recorded COUNT_LINK_DATA_PACKETS as RETRANSMITTED_PACKETS
```

```
PROCESS NAME:          IF STRT, STACK, OR DATA_PACKET THEN START TIMER
PROCESS NUMBER:        1.8.1
PROCESS DESCRIPTION:
IF Input = QSYNC_ENQ/DLE_PACKET, STRT_PACKET, or STACK_PACKET then
     Start TIMER
     Send LINK_AND_MODEM_CONTROL = ENABLE_LINK
```

```
     Send a block to the driver through the use of the TRANSMIT_A_BLOCK_
          COMMAND, it will be outputted as PRIMARY_OUTGOING_BIT_STREAM_TO_
          DRIVER
ELSEIF Input = PRIMARY_TO_SECONDARY_DATA_PACKET then
     Start TIMER
     Send LINK_AND_MODEM_CONTROL = ENABLE LINK
     Send a block t the driver through the use of the TRANSMIT_A_BLOCK_
          COMMAND, it will be outputted as PRIMARY_OUTGOING_BIT_STREAM_TO_
          DRIVER
     Send PRIMARY_TO_SECONDARY_DATA_PACKET NUM value to reset TIMER process
ELSEIF Input = RESET_SIGNAL then
     Reset the TIMER
ELSEIF TIMER times out the
     Output TIMEOUT
ELSE Input = STOP_LINK then
     Send last PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER packet with an ENABLE_
          LINK then
     Send a DISABLE_LINK to the driver


PROCESS NAME:          RESET TIMER
PROCESS NUMBER:        1.8.2
PROCESS DESCRIPTION:
IF Input new A_FILE value then
     Compare with last recorded NUM value
     IF equal then
          Reset TIMER
     ELSE Null
ELSEIF Input is RECEIVED_STRT_PACKET then
     Reset TIMER
ELSE Input is RECEIVED_STACK_PACKET then
     Reset TIMER


PROCESS NAME:          COUNT ERRORS (TIME)
PROCESS NUMBER:        1.8.3
PROCESS DESCRIPTION:
EACH time a TIMEOUT occurs then
     Increment COUNTER
     IF COUNTER = 7 then
          Output PERSISTENT_ERROR
          Reset  COUNTER
     ELSE Null


PROCESS NAME:          IDENTIFY TIMEOUT PACKET
PROCESS NUMBER:        1.8.4
PROCESS DESCRIPTION:
```

```
IF TIMEOUT due to DATA_PACKET then
     Output TIMEOUT_DATA
ELSEIF TIMEOUT due to STRT_PACKET then
     Output TIMEOUT_STRT
ELSE output TIMEOUT_STACK
```

PROCESS NAME: TRANSMIT BIT STREAMS
PROCESS NUMBER: 2.0
PROCESS DESCRIPTION:
USE the PHYSICAL LEVEL PROTOCOL to transmit the PRIMARY_OUTGOING_BIT_STREAM_TO_DRIVER to the SECONDARY NODE and the SECONDARY_OUTGOING_BIT_STREAM_TO_DRIVER to the PRIMARY NODE.

PROCESS NAME: EXECUTE HDLC PROTOCOL AT SECONDARY NODE
PROCESS NUMBER: 3.0
PROCESS DESCRIPTION:
USE the LINK LEVEL PROTOCOL to transmit an error free SECONDARY_INCOMING_BIT_STREAM_FROM_DRIVER to the SECONDARY NODE and to transmit an error free RECEIVED_SECONDARY_NODE_TRANSPORT_PACKET to the PRIMARY NODE.

Appendix D

Electronic Warfare Network (EWNET) Profile

This appendix contains a compilation of the EWNET hardware and software design features using a format required for use by the Digital Equipment Corporation.

Contents

Electronic Warfare Network (EWNET) Profile

for

Robins AFB, Georgia

18-DEC-81

Revision 1

Developed by: Robert H. Stokes

Approved by:

Network Profile

"Customer Information"

Robins AFB

c/o Mr. Joe Black

WR-ALC / MMRR

Robins Air Force Base, Georgia 31098

Customer Network Manager: Mr. Roger Boan

Location: Robins Air Force Base

"Digital Account Management"

Account Manager:	Steven Jones
Software Specialist:	Jim Bell
Office:	ATD 2
District Software Manager:	Ed Converse

"Application Information"

The network will be used in an application doing task-to-task communication, peripheral sharing, and file transfers as described in AFIT Thesis "Design of a Local Computer Network for the Robins AFB, Electronic Warfare Division Engineering Branch Laboratory" by Robert H. Stokes. There exist special modifications to DECnet in this application which constitutes overall functionality constraints and critical factors.

"Potential Problem Areas"

Does proposed network cross area, region, district, or branch boundaries?

NO

Are any phased installation / warranty problems foreseen?

NO

Any network-related special purpose hardware or software involved?

YES

Fiber optic link and modems for high speed and bandwidth

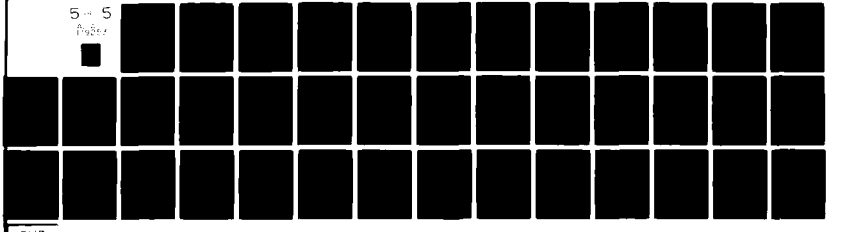
"Overall Description"

Digital Supplied Customer Supported

Responsible Product Line:	GSG
Other Product Lines:	None
Number of Nodes:	14 Phase II
	3 Phase III
Maximum number of links on any node:	7
Operating systems:	VMS
	RSX-11M

AD-A119 253 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 9/2
DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONI--ETC(U)
DEC 81 R H STOKES
UNCLASSIFIED AFIT/GCS/EE/81D-16 NL

5 - 5
AD-A119 253



END
DATE FILMED
10-82
DTIC

"DECnet Functionality to be used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing (Satellite)	X			
Adaptive Routing	X			

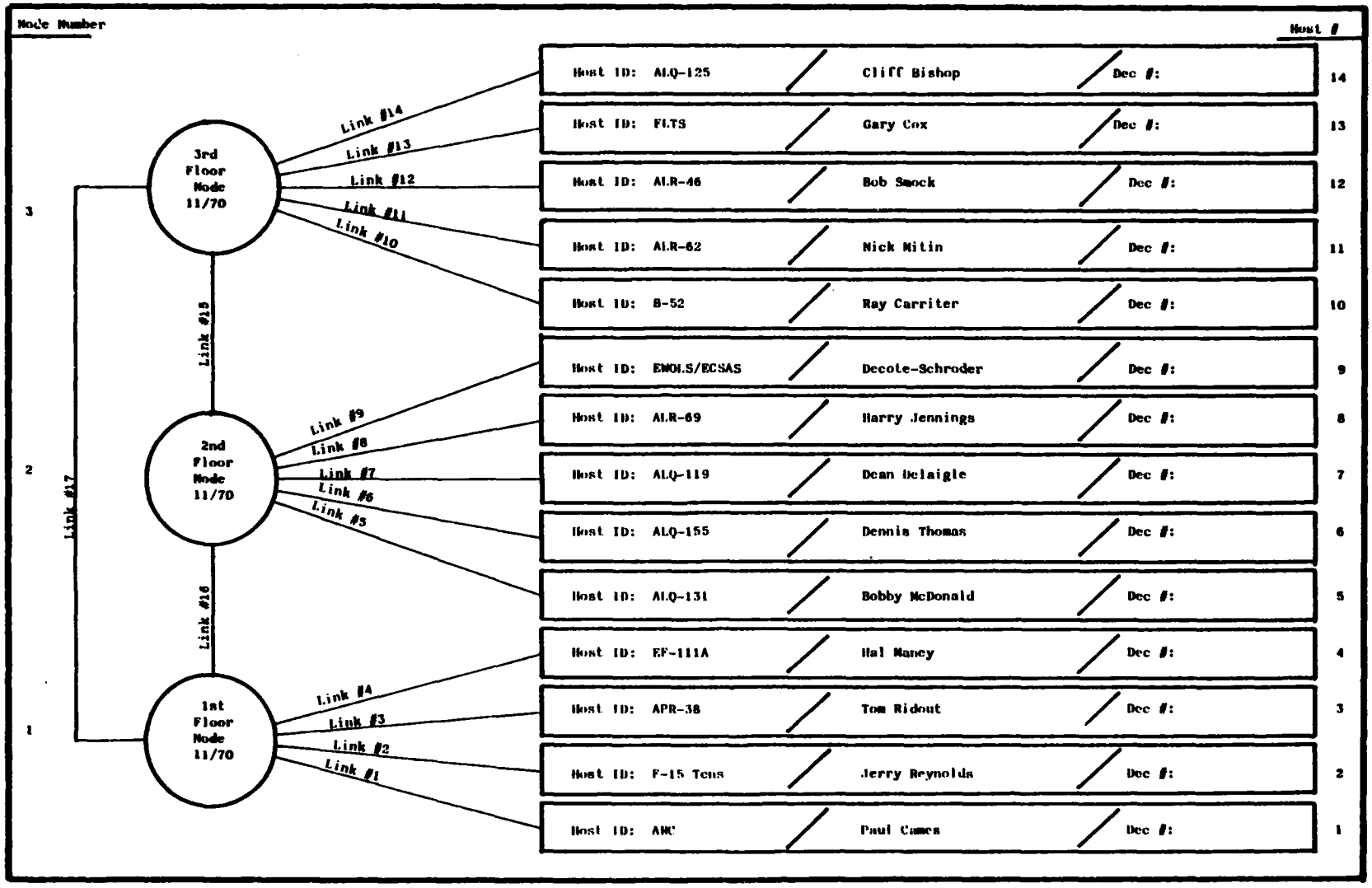


Figure D-1 Initial Ewnet Configuration

Link Description

Link Number: 1

Node Name: First Floor Node	Linked to Node Name: ARC
Interface: DMC11-AR DMC11-MA	Interface: DMC11-AR DMC11-MA
Bootstrap:	Bootstrap:
Operating System: RSX-11M	Operating System: RSX-11M
Line Speed: 1M Baud	

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 Bytes
Expected Line Utilization:	50%

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 2

Node Name:	First Floor Node	Linked to Node Name:	F-15 TEWS
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50%

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 3

Node Name:	First Floor Node	Linked to Node Name:	APR-38
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-toTask	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 4

Node Name:	First Floor Node	Linked to Node Name:	EF-111A
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

link Number: 5

Node Name:	Second Floor Node	Linked to Node Name:	ALQ-131
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 6

Node Name: Second Floor Node	Linked to	Node Name: ALQ-155
Interface: DMC11-AR DMC11-MA		Interface: DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:
Operating System: RSX-11M		Operating System: VMS
Link Speed: 1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number 7

Node Name:	Second Floor Node	Linked to **Node Name:**	ALQ-119
Interface:	DMC11-AR DMC11-MA	**Interface:**	DMC11-AL DMC11-MA
Bootstrap:		**Bootstrap:**	
Operating System:	RSX-11M	**Operating System:**	VMS

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 8

Node Name:	Second Floor Node	Linked to Node Name:	ALR-69
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 9

Node Name:	Second Floor Node	Linked to Node Name:	EWOLS / ECSAS
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AL DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 10

Node Name:	Third Floor Node	Linked to Node Name:	B-52
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AR DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	RSX-11M
Link Speed:	1M baud		

"Link Characteristics"

Serial	Synchronous	Full-Duplex	Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 11

Node Name: Third Floor Node	Linked to	Node Name: ALR-62	
Interface: DMC11-AR DMC11-MA		Interface: DMC11-AL DMC11-MA	
Bootstrap:		Bootstrap:	
Operating System: RSX-11M		Operating System: VMS	
Link Speed: 1M baud			

"Link Characteristics"

Serial	Synchronous	Full-Duplex	Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 12

Node Name:	Third Floor Node	Linked to Node Name:	ALR-62
Interface:	DMC11-AR	Interface:	DMC11-AL
	DMC11-MA		DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	VMS
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Descriptions

Link Number: 13

Node Name:	Third Floor Node	Linked to Node Name:	FLTS
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AR DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	RSX-11M
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 14

Node Name: Third Floor Node	Linked to	Node Name: ALQ-125	
Interface: DMC11-AR DMC11-MA		Interface: DMC11-AR DMC11-MA	
Bootstrap:		Bootstrap:	
Operating System: RSX-11M		Operating System: RSX-11M	
Link Speed: 1M baud			

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	50 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing		X		X

Link Description

Link Number: 15

Node Name: Third Floor Node	Linked to	Node Name: Second Floor Node
Interface: DMC11-AR DMC11-MA		Interface: DMC11-AR DMC11-MA
Bootstrap:		Bootstrap:
Operating System: RSX-11M		Operating System: RSX-11M
Link Speed: 1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	100 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto_Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing			X	

Link Description

Link Number: 16

Node Name: Second Floor Node	Linked to	Node Name: First Floor Node
Interface: DMC11-AR DMC11-MA		Interface: DMC11-AR DMC11-MA
Bootstrap:		Bootstrap:
Operating System: RSX-11M		Operating System: RSX-11M
Link Speed: 1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	100 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down_Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing			X	

Link Description

Link Number: 17

Node Name:	First Floor Node	Linked to Node Name:	Third Floor Node
Interface:	DMC11-AR DMC11-MA	Interface:	DMC11-AR DMC11-MA
Bootstrap:		Bootstrap:	
Operating System:	RSX-11M	Operating System:	RSX-11M
Link Speed:	1M baud		

"Link Characteristics"

Serial Synchronous Full-Duplex Local

"Expected Data Throughput"

Average:	1M baud
Peak:	1M baud
Size of Average Data Message:	512 bytes
Expected Line Utilization:	100 %

"DECnet Functionality to be Used"

	Present		Future	
	YES	NO	YES	NO
Task-to-Task	X			
File Transfer	X			
File Access	X			
Batch / Command File Submission	X			
Batch / Command File Execution	X			
Down-Line System Loading			X	
Down-Line Task Loading			X	
Network Command Terminals (Homogeneous)		X		X
Network Command Terminals (Hetergeneous)			X	
Auto-Answer Telephone		X		X
Auto-Dial Telephone		X		X
Multidrop DDCMP		X		X
Routing	X			
Adaptive Routing			X	

Host Description

Host Name: ARC
Host Number: 1
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: RSX-11M
DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/34
Main Memory: 1024 K bytes MOS

Terminals:	Number	Type	Speeds
	3	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	TJE-16
1	RJM-02
1	LP-11

Total Number of Links? 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 1.

Host Description

Host Name: F-15 TEWS — Host Number: 2
Location: Robins AFB
Expected Date On-Line: Jan 1982

Operating System: VMS — DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	YES
D/A Conversion?	YES
Graphics?	YES
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	YES

CPU: VAX-11/780 — Main Memory: 1024 K bytes MOS

Terminals:

Number	Type	Speeds
8	VT-100	9600 baud
1	VT-125	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
2	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet? 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 2.

Host Description

Host Name: APR-38 Host Number: 3
Location: Robins AFB
Expected Date On-Line: Sept 1982

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	YES
D/A Conversion?	YES
Graphics?	YES
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 Main Memory: 1024 K bytes MOS

Terminals:

Number	Type	Speeds
8	VT-100	9600 baud
1	VT-125	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
2	REM-03
1	LA-120

Total Number of Links? 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet? 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 3.

Host Description

Host Name: EF-111A — Host Number: 4
Location: Robins AFB
Expected Date On-Line: Dec 1982

Operating System: VMS — DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	YES
D/A Conversion?	YES
Graphics?	YES
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	YES

CPU: VAX-11/780 — Main Memory: 1024 K bytes MOS

Terminals:

Number	Type	Speeds
4	VT-100	9600 baud
1	VT-125	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
2	TEE-16
2	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Seconds)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 4.

Host Description

Host Name: ALQ-131 — Host Number: 5
Location: Robins AFB
Expected Date On-Line: 15-Jun-81

Operating System: VMS — DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 — Main Memory: 1024 K bytes MOS

Terminals:

Number	Type	Speeds
8	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
1	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 5.

Host Description

Host Name: ALQ-155 Host Number: 6
Location: Robins AFB
Expected Date On-Line: 15-Jun-81

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 Main Memory: 1024 K bytes MOS

Terminals: Number	Type	Speeds
6	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
1	REM-03
1	LA-120

Total Number of Links? 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 6.

Host Description

Host Name: ALQ-119 Host Number: 7
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 Main Memory: 1024 K bytes MOS

Terminals: Number	Type	Speeds
4	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
1	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under *Link Description 7.*

Host Description

Host Name: ALR-69 Host Number: 8
Location: Robins AFB
Expected Date On-Line: Jan 1982

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 Main Memory: 1024 K bytes MOS

Terminals: Number	Type	Speeds
4	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	LP-11
1	TEE-16
2	REM-03

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 8.

Host Description

Host Name: EWOLS/ECSAS — Host Number: 9
Location: Robins AFB
Expected Date On-Line: 15-Jun-81

Operating System: VMS — DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 — Main Memory: 1024 K bytes MOS

Terminals: Number	Type	Speeds
22	VT-100	9600 baud

Other Peripherals:

Number	Type
1	LP-11
6	DR-11
1	TEE-16
2	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This Host will use DECnet for all uses listed as applicable under Link Description 9.

Host Description

Host Name: B-52 Host Number: 10
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: RSX-11M DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/34 Main Memory: 1024 K bytes MOS

Terminals:	Number	Type	Speeds
	2	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TJE-16
1	RJM-02
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 10.

Host Description

Host Name: ALR-62 Host Number: 11
Location: Robins AFB
Expected Date On-Line: Dec 1981

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: VAX-11/780 Main Memory: 1024 K bytes MOS

Terminals: Number	Type	Speeds
6	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
1	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 11.

Host Description

Host Name: ALR-46 Host Number: 12
Location: Robins AFB
Expected Date On-Line: Feb 1982

Operating System: VMS DECnet Version: 1.3
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	YES
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

Terminals:

Number	Type	Speeds
6	VT-100	9600 baud
2	VT-125	9600 baud
1	TEK-4027	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LP-11
1	TEE-16
2	REM-03
1	LA-120

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 12.

Host Description

Host Name: FLTS — Host Number: 13
Location: Robins AFB
Expected Date On-Line: Sept 1982

Operating System: RSX-11M — DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/34 — Main Memory: 1024 K bytes MOS

Terminals:	Number	Type	Speeds
	4	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	TJE-16
1	RJM-02
1	LP-11

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 13.

Host Description

Host Name: ALQ-125 Host Number: 14
Location: Robins AFB
Expected Date On-Line: Nov 1981

Operating System: RSX-11M DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/34 Main Memory: 1024 K bytes MOS

Terminals:	Number	Type	Speeds
	4	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	TJE-16
1	RJM-02
1	LP-11

Total Number of Links: 1

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 50 %

"Host Application"

This host will use DECnet for all uses listed as applicable under Link Description 14.

Node Description

Node Name: First Floor Node — Node Number: 1
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: RSX-11M — DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/70 — Main Memory: 2048 K bytes MOS

Terminals: Number	Type	Speeds
2	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	LP-11
3	RWM-02

Total Number of Links: 3

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 100 %

"Node Application"

This node will use DECnet for all uses listed as applicable under Link Description 15, plus act as an EWNET Routing node.

Node Description

Node Name: Second Floor Node — Node Number: 2
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: RSX-11M — DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/70 — Main Memory: 2048 K bytes MOS

Terminals:

Number	Type	Speeds
2	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	LP-11
3	RWM-03

Total Number of Links: 3

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 100 %

"Node Application"

This node will use DECnet for all uses listed as applicable under Link Description 15, plus act as an EWNET Routing node.

Node Description

Node Name: Third Floor Node — Node Number: 3
Location: Robins AFB
Expected Date On-Line: 1983

Operating System: RSX-11M — DECnet Version: 3.0
Languages Supported: Fortran and Pascal
Other Layered S/W Products:

A/D Scanning?	NO
D/A Conversion?	NO
Graphics?	NO
2780,3780,HASP,3271,SNA?	NO
Other Interrupt Intensive S/W?	NO

CPU: PDP-11/70 — Main Memory: 2048 K bytes MOS

Terminals: Number	Type	Speeds
2	VT-100	9600 baud

Other Peripherals:

Number	Type
1	DR-11
1	LA-120
1	LP-11
3	RWM-03

Total Number of Links: 3

Total Throughput: (Bits/Second)

	IN	OUT
Average:	125 K	125 K
Peak:	125 K	125 K

Expected CPU Utilization by DECnet: 100 %

"Node Application"

This node will use DECnet for all uses listed as applicable under Link Description 15, plus act as an EWNET Routing node.

Peripheral Descriptions (Ref. 3:18-20)

VT-100	=	Standard CRT with Advance Video
VT-125	=	Graphics CRT
TEK-4027	=	Graphics CRT
LA-120	=	180 Character Hard-Copy Terminal
RJM-02	=	67 M byte Freestanding Disc, Controller, and Drive PDP-11/34
RWM-03	=	67 M byte Freestanding Disc, Controller, and Drive PDP-11/70
REM-03	=	67 M byte Freestanding Disc, Controller, and Drive VAX-11/780
TEE-16	=	800/1600 BPI, 45 IPS, 9 Track Mag Tape Controller and Drive. VAX-11/780
TJE-16	=	800/1600 BPI, 45 IPS, 9 Track Mag Tape Controller and Drive. PDP-11/70 and PDP-11/34
LP-11	=	600 LPM Line Printer
DR-11	=	DMA Unibus Interface

Vita

Mr. Robert H. Stokes was born on November 2, 1948, in Athens, Tennessee. In 1967, he graduated from McMinn Central High School in Englewood, Tennessee. He served with the United States Air Force as a communications specialist in Europe and was honorably discharged in 1972. He attended the Tennessee Technology University from which he received a Bachelor of Science in Electrical Engineering degree in 1975. Following graduation, he accepted employment with the Electronic Warfare Division of the United States Air Force as a civilian electronic engineer working on the F-15 and F-4E Tactical Electronic Warfare Systems. Between June 1977, and December 1978, he attended Georgia College and in December of 1978, he received a Master of Science in Adminstration degree. He entered the Air Force Institute of Technology in June 1980.

Perminate address: Highway 411, South
Etowah, Tennessee 37331

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/81D-16	2. GOVT ACCESSION NO. AD-A112 953	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* DESIGN OF A LOCAL COMPUTER NETWORK FOR THE ROBINS AFB ELECTRONIC WARFARE DIVISION ENGINEERING BRANCH LABORATORY		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert H. Stokes, Civ, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, OH 45433		12. REPORT DATE December 1981 13. NUMBER OF PAGES 423
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

8 SEP 1982

Frederic C. Lynch, Major, USAF
Director of Public Affairs

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Local Computer Network
X.25 Protocol
Computer Interfaces
DECNET Protocol

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

See reverse

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Continuation of Block 20

A local computer for the Electronic Warfare Division Engineering Branch Laboratory was designed around a commercially available and supportable network configuration. The requirements for this network were specified by interviewing the engineers associated with the Engineering Branch Laboratory and then translating the functional requirements into a detailed set of hardware and software system requirements. Structured Analysis was used to produce a structured specification for application, transport, network, and data link protocol level requirements. Digital Equipment Corporation's 'DECnet" Phase II and Phase III network configurations were combined together to form this unique Electronic Warfare Network (EWNET). The node network uses a loop topology with a star of up to seven hosts connected to each node. The nodes are implemented using Digital Equipment Corporation's PDP-11/70 computers. Initially, the network will include fourteen Integrated Support Station computers which are either Digital VAX-11/780s or Digital PDP-11/34s. These computers will be connected to the nodes using duplex fibier optic links supporting transmission rates up to 1 Mbs. The Phase II DECnet protocol was selected to provide the file transfer and data transport protocols in conjunction with a basic routing algorithm at each host, while the Phase III DECnet protocol was selected to provide these functions at a higher level in the nodes. The selection of the above topology in conjunction with the described protocol structure keeps any one host from degrading the network if the host should fail. All Integrated Support Station common off-line functions, common databases, and commonly used support software tools are hosted on the node computer for easy, universal access, allowing for a degree of standardization.