1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

CANADIAN THESES
ON MICROFICHE

THÈSES CANADIENNES
SUR MICROFICHE

0-315-22199-2

NAME OF AUTHOR/NOM DE L'AUTEUR

José RAMIRO MONTALCERO

TITLE OF THESIS/TITRE DE LA THÈSE

Excursion - Workload Modelling of a
Transport Protocol

UNIVERSITY/UNIVERSITÉ

Carleton University

DEGREE FOR WHICH THESIS WAS PRESENTED/

GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE

M.C.S.

YEAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADE

1984

NAME OF SUPERVISOR/NOM DU DIRECTEUR DE THÈSE

C. MURRAY Woods

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies
of the film.

The author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et
de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la
thèse ni de longs extraits de celle-ci ne doivent être imprimés
ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/DATE

16 April 84

SIGNED/SIGNÉ

José Ramiro Montalero

PERMANENT ADDRESS/RÉSIDENCE FIXE

12 Calle "A" 0-66 Zona 1

GUATEMALA, GUATEMALA

CENTRAL AMERICA

EXECUTION-WORKLOAD MODELLING OF A

TRANSPORT PROTOCOL

By

© Jose Ramiro Montealegre Villacorta

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for
the degree of
Master of Computer Science

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

Ottawa, Ontario

March, 1964.

LICENCE TO CARLETON UNIVERSITY

In the interests of facilitating research by others at this institution and elsewhere, I hereby grant a licence to:

CARLETON UNIVERSITY

to make copies of my thesis:

Simulation - WEIRAD MODELLING OF A
TRANSPORT PROTOCOL

or substantial parts thereof, the copyright which is invested in me, provided that the licence is subject to the following conditions:

Only singled copies shall be made or authorized to be made at any one time, and only in response to a written request from the library of any university or similar institution on its own behalf or on behalf of one of its users.

This licence shall continue for the full term of the copyright, or for so long as may be legally permitted.

The Universal Copyright Notice shall appear on the title page of all copies made under the authority of this licence.

This licence does not permit the sale of authorized copies at a profit, but does permit the collection by the institution or institutions concerned of charges covering actual costs.

All copies under the authority of this licence shall bear a statement to the effect that the copy in question "is being made available in this form by the authority of the copyright owner solely for the purpose of private study and research and may not be copied or reproduced except as permitted by the copyright laws without written authority from the copyright owner."

The foregoing shall in no way preclude the granting by the author of a licence to the National Library of Canada to reproduce the thesis and to lend or to sell copies of the same. For this purpose it shall also be permissible for Carleton University to submit the abovementioned thesis to the National Library of Canada:

Signature of Witness

16 April 84 M.C.S.
Date Degree

[Signature]
Signature of Student

Systems & Computer Eng.
Department of Student

The undersigned, recommends to the Faculty of
Graduate Studies and Research acceptance of the thesis

"Execution-workload modelling of a
Transport Protocol"

submitted by Jose Ramiro Montealegre Villacorta in partial
fulfillment of the requirements for the degree of
Master of Computer Science.

C.M. Lindsay
Thesis Supervisor

B. Pagnier
Chairman, Department of Systems
and Computer Engineering

Carleton University
Faculty of Engineering

March 15, 1984

IN MEMORY OF MY FATHER,
TO MY MOTHER,
SISTER AND BROTHER...

...And to those who believe that
"The whole fun of living is
trying to do things better".

EXECUTION WORKLOAD MODELLING OF A TRANSPORT PROTOCOL

ABSTRACT

The usefulness of a distributed system depends on its throughput and delay performance. These characteristics are most sensitive to resource allocation schemes, overhead, and traffic statistics. By identifying the most critical influencing components of the protocol it is possible to avoid bottlenecks. Such bottlenecks are, usually, due to implementation decisions rather than to the protocol rules. It is desirable to predict performance effects such as processor saturation, buffer management techniques and allocation of protocol functions at the design stage.

The problem of designing high performance computer communication protocols is that of identifying and modelling the requirements of the protocol, a set of environmental constraints, and using estimating rules (to estimate the system performance, i.e. cost-effectiveness) be able to generate a set of possible designs in accordance with the environment. The performance behavior of the system and comparisons among different designs which satisfy the requirements could then be predicted during design stages. This thesis addresses the need of a careful assessment of the data communication protocol components, and goes on to describe, in a systematic way, the construction process of

ABSTRACT

11

an execution load model for computer communication protocols to propose and compare implementation designs. In particular the data transfer phase of an OSI class 4 Transport protocol has been modelled in this study. That means that all levels are assumed initialized in the active channel state and all virtual circuits are in data transfer mode.

ACKNOWLEDGMENTS

Special mention goes to Professor Murray Woodside, whose constant encouragement, ideas and assistance helped me to achieve this goal.

I am very grateful to my parents, sister and brother for their full support through the space and time...

Lastly, I would like to thank those persons in the Department of Systems and Computer Engineering at Carleton University, in the Department of Communications of Canada, and all my friends that in one way or other encouraged me during the course of this thesis. I specially wish to mention Suzanne Chouinard, Lucrecia Chaluleu, Ana Bella Castillo, Fred Kaudel, Ivan Taylor and Chris Griffiths for their cooperation, patience and tolerance during the hours devoted to my studies.

TABLE OF CONTENTS

	Page
Abstract	i
Acknowledgment	iii
Table of Contents	iv
List of Figures	vi
List of Tables	viii
Nomenclature	ix
 Chapter 1 Introduction	
1.0 Motivation	1-1
1.1 Overview	1-2
1.2 Workload Model	1-3
 Chapter 2 Description of an ISO Transport Protocol	
2.0 Introduction	2-1
2.1 Architectural Model	2-1
2.2 Purpose of a Transport Protocol	2-3
2.2.1 Establishment Phase	2-3
2.2.2 Data Transfer Phase	2-4
2.2.3 Termination Phase	2-4
2.3 Classes and Options	2-5
2.4 Services Assumed from the Network Layer	2-6
 Chapter 3 Transport Requirements	
3.0 Introduction	3-1
3.1 Internal Function Required	3-2
3.2 Transport Layer's Data Transfer Phase	
Transitions using FSMs	3-3
3.2.1 Connection Data Transfer	3-3
3.2.2 Expedited Data Transfer	3-5
3.2.3 Discussion	3-5
 Chapter 4 Design Decisions and Parameters	
4.0 Introduction	4-1
4.1 Set of System Designs	4-2
4.2 Operational Environment Design Issues	4-3
4.2.1 Computer System Configuration	4-4
4.2.1.1 Organization and Synchronization of Protocol functional Units	4-4

TABLE OF CONTENTS

v

4.2.1.2	Integrating the Protocol to the system	4-10
4.2.2	Traffic Pattern	4-11
4.2.3	Network Services Available	4-12
4.2.3.1	Network Reliability	4-12
4.2.3.2	Availability of Network Services	4-12
4.2.4	Availability of Special Hardware	4-14
4.3	One End Protocol Interfaces Design Issues	4-16
4.4	Multiplicity of Connections Design Issues	4-16
4.4.1	Matching End_to_End Paths Vs. Multiplexing	4-16
4.4.2	Separation of Error Recovery and Multiplexing	4-17
4.5	Buffer Management	4-18
4.6	Other Issues	4-21

Chapter 5

5.0	Introduction	5-1
5.1	Workload Components	5-1
5.2	Workload Estimation	5-2
5.2.1	Organization of the Software	5-3
5.2.1.1	Sender	5-3
5.2.1.2	Receiver	5-5
5.2.2	Experiment Design and Results	5-7
5.2.2.1	Outgoing Traffic	5-7
5.2.2.2	Incoming Traffic	5-14

Chapter 6 Protocol Design Cases

6.0	Introduction	6-1
6.1	Buffer Management Techniques Analysis	6-1
6.1.1	Processing Time for Handling Outgoing messages	6-1
6.1.2	Fragmentation Overhead	6-5
6.2	Throughput Performance	6-6
6.2.1	Mean Execution Time at Level 4	6-8
6.2.2	Throughput Capacity -Long Haul Vs. Local Network-	6-10
6.3	Transit Time Performance	6-13
6.4	Comparison of Class 4 and Class 2	6-14

Chapter 7 Conclusions

7-1

Appendix A: Definitions

A-1

Appendix B: Transport Service Primitives

B-1

Appendix C: Execution Time Derivations for Procedure Oriented Intercommunication Primitives

C-1

References

R-1

LIST OF FIGURES

FIGURE		PAGE
1-1	A Systematic and Incremental Protocol Model Construction and Validation Cycle	1-5
2-1	Three Layers Architecture Model	2-8
3-1	Graphs Notation	3-7
3-2	Functional Components of an ISO Transport Protocol	3-8
3-3	Class 4 Transport Data Transmission FSM	3-9
3-4	Class 4 Transport Data Reception FSM	3-10
3-5	Transport Class 4 Expedited Data Transfer FSM	3-10
4-1	Functional Units Organization Alternatives	4-22
4-2	Connection Binding: Matching End_to_End Paths	4-23
4-3	Connection Binding: Multiplexing	4-23
4-4	Logical Components of the "Bottom Level Recovery" Method	4-24
4-5	Queueing Model for the "Bottom Level Recovery" Method	4-24
4-6	Logical Components of the "Demultiplexed" Method	4-25
4-7	Queueing Model for the "Demultiplexed" Method	4-25
4-8	Four Buffer Management Implementation Alternatives	4-26
5-1	Four Phases Model of a Computer Communication Protocol	5-17
5-2	Queueing Model for the Transport Packet Execution	5-17
5-3	SENDER Process Modular Representation	5-18
5-4	RECEIVER Process Modular Representation	5-19
5-5	Execution Time Distribution for Outgoing Traffic (TSDU size = 128 bytes)	5-20

LIST OF FIGURES

vii

5-6	Execution Time Distribution per Routine for Sending 10,000 128-byte TSDUs	5-21
5-7	Execution Time Distribution per Routine for Sending a 2048-byte TSDU	5-21
6-1	Buffer Management Techniques: Processing Time for Handling an Outgoing TSDU	6-15
6-2	Buffer Management Processing Time to Send a 1024 byte-TSDU under Different Fragmentation Schemes.	6-15
6-3	Mean Execution Time per TSDU at the Transmitter (Ts) for Different Levels of Fragmentation	6-16
6-4	Mean Execution Time at Level 4, to Transfer 1,000 512 byte-TSDUs under different acknowledgment policies.	6-16
6-5	Maximum Throughput Capacity -Long Haul Vs. Local Net- Transferring 64-byte TPDUs	6-17
6-6	Maximum Throughput Capacity -Long Haul Vs. Local Net- Transferring 512-byte TPDUs	6-17
6-7	TRANSIT Time Components for the Transfer of a 64-Byte TSDU	6-18
6-8	TRANSIT Time Components for the Transfer of a 512-Byte TSDU	6-18
6-9	Mean Execution Time for a 512-Byte TSDU at Level 4, for Class 2 and Class 4	6-19

LIST OF TABLES

TABLE		PAGE
4-1	Typical Callable Procedure Synchronization Operations	4-6
4-2	Typical Procedure-Oriented Synchronization Operations	4-8
5-1	Protocol Processing Time (First Trial: Sending)	5-8
5-2	Time Spent per Protocol Function (First Trial: Sending)	5-8
5-3	Protocol Processing Time (Second Trial: Sending)	5-12
5-4	Time Spent per Protocol Function (Second Trial: Sending)	5-12
5-5	Time Spent per Protocol Function (First Trial: Receiving)	5-14
5-6	Time Spent per Protocol Function (Second Trial: Receiving)	5-16
6-1	Buffer Management Techniques: Processing Time Per Message Without Fragmentation	6-4
6-2	Buffer Management Techniques: Processing Time Per Message With Fragmentation	6-7

NOMENCLATURE

N.1 SYMBOLS AND ABBREVIATIONS:

N.1.1 DATA UNITS:

TPDU	Transport protocol data unit
TSDU	Transport service data unit
NSDU	Network service data unit

N.1.2 TYPES OF TRANSPORT PROTOCOL DATA UNITS:

CR_TPDU	Connection request TPDU
CC_TPDU	Connection confirm TPDU
DR_TPDU	Disconnect request TPDU
DC_TPDU	Disconnect confirm TPDU
DT_TPDU	Data TPDU
ED_TPDU	Expedited data TPDU
AK_TPDU	Data acknowledge TPDU
EA_TPDU	Expedited acknowledge TPDU
RJ_TPDU	Reject TPDU
ER_TPDU	Error TPDU

NOMENCLATURE

N.1.3 TPDU FIELDS:

LI	Length indicator (field)
CDT	Credit (field)
TSAP-ID	Transport service access point identifier (field)
DST-REF	Destination reference (field)
SRC-REF	Source reference (field)
EOT	End of TSDU mark
TPDU-NR	DT TPDU number (field)
ED-TPDU-NR	ED TPDU number (field)
YR-TY-NR	Sequence number response (field)

N.1.3 MISCELLANEOUS:

TS-USER	Transport service user
TSAP	Transport service access point
NS-PROVIDER	Network service provider

N.2 MATHEMATICAL SYMBOLS USED:

λ	= Throughput
AK	= Average number of Acknowledgments per TPDU
B	= Bytes per TPDU
B1	= Bytes per TSDU
Delay	= Average delay time for the Network Service
H1	= Bytes per AK_TPDU
H2	= Bytes per RJ_TPDU

K	=	Maximum number of transmissions
L	=	Probability of a fail transmission
M	=	Average number of TPDUs per TPDU
R	=	Average number of Retransmissions per TPDU
RJ	=	Average number of Rejections per TPDU
Tc	=	Network Service Control_TPDU transit delay (in one direction)
Tn	=	Network Service Data_TPDU transit delay (in one direction)
Tr	=	Mean Execution Time at the receiver
Ts	=	Mean Execution Time at the transmi- tter
W	=	Window Size

(See also definitions in Appendix A)

CHAPTER 1

INTRODUCTION

1.0 MOTIVATION:

There is a growing literature on protocol performance, analyzing the effect on throughput and delay of various protocol rules and parameters. For example for flow control rules, Gerla and Kleinrock surveyed flow control problems in [GERLA 80]. In [REISER79] is described a queueing network solution method suitable for solving a model of a network with many flow-controlled connections. In the local networks field, the literature is flooded by articles describing analytical works dealing with performance comparisons [TOBAGI79], [BUX 80] of the various proposed channel partition methods. However, little attention has been paid to the execution performance of protocol implementation, meaning the effect of processing load and synchronization delays within processors at one end of the connection. For example, the 97% channel utilization capability of the Ethernet reported in [SHOCH 80] completely ignores the processing time at the network interface, the interaction between the channel access and the higher level protocol, buffer management and the nodes' speed mismatch. [WATSON80] reports on a simulation of two CSMA/CD type networks using different protocol parameters and concludes that network performance is strongly dependent on these factors, sometimes reducing utilization to less than 50%. This thesis is concerned with the execution workload generated by a protocol implementation. By execution workload is meant the seconds of processing for the various operations, which in turn determine delays in processing and possible processing bottlenecks.

1.1 OVERVIEW

The first requisite on a protocol implementation is to meet the specifications, for example the ISO Transport specifications. Beyond this, an implementer may have many other goals related to the space and speed for instance. The specifications do not guide or constrain him on these matters. This thesis describes execution workload by a model which gives the processing load as a function of the various factors the designer must consider. Our ultimate aim is a model which can be used to predict the processing load directly from a sample design. The model takes into account:

- Throughput Bottlenecks,
- System response with respect to unbalanced load and dynamic overload,
- Identification of the most critical system parameters, and
- Optimization of system parameters, in relation to the environment.

The sequence of topics in the thesis is outlined in Figure 1-1. Chapter 2, is devoted to the understanding of the data transfer phase of an OSI transport protocol. Chapter 3 describes the requirements, and Chapter 4 the major design issues for the transport protocol. Chapter 5 describes an experiment, and in Chapter 6 the workload estimation of the experiment is used to compare different design alternatives, and their side effects, under different operational environments.

The author's contributions are (1) the identification of the hardware/software design issues, (Chapter 4); (2) the determination of the processing times of the essential protocol functions (Chapter 5); the use of these processing times for bounding performance (Chapter 6).

1.2 WORKLOAD MODEL:

The workload model (W) is defined as a function:

$$W = F(R, E, D, U)$$

Where the arguments:

R = A set of requirements for system actions. Such as the requirement to transmit voice along with data.

E = Operational environment. For instance Rate of errors.

D = A set of design elements, which reflects the particular environment. For example the use of checksum, traffic priority classes schemes.

U = Set of protocol parameters, such as window size, timeout values.

INTRODUCTION

1-4

The workload function is viewed as the requirements of the protocol (R) selected according to the particular environment (E), and reflected in the design elements (D). Notice that the workload function, as described above is incompletely defined until the protocol parameter set, U, is specified. However in the selection of U the environment constraints must be taken into account. The selection of U will result in the selection of exactly one protocol workload model for the particular environment with a specific performance.

FIGURE 1-1. A Systematic and Incremental Protocol Model Construction and Validation Cycle.

CHAPTER 2

DESCRIPTION OF AN I.S.O TRANSPORT PROTOCOL

2.0 INTRODUCTION

The first step in the systematic construction and validation of the Transport protocol model consists of finding the functional elements of the protocol without any performance measuring and statistics gathering facilities. This chapter is devoted to the understanding phase of an I.S.O. Transport Protocol.

2.1 ARCHITECTURAL MODEL

It is possible to use several models to represent, in a hierarchical way, a computer network, as shown in [GREEN 79] and [POUZIN79]. The idea behind a model is to provide an abstract implementation that notifies the principles of the system architecture. A model reveals the structure and structural interaction in an environment free of distraction (and confusing) implementation detail. Specially in the development of complex standards, such as transport, it is useful to follow a guideline which abstractly models the protocol. Such a guideline exists for the transport in the form of the Open System Interconnection (OSI) Reference Model [ISO 79]. The OSI model was developed by the International Organization for Standardization, precisely as an aid in developing standard protocols. This model divides the computer network functions and services into seven protocol layers which collectively address network protocol modelling. Each host site on a network supports the seven layers. Such

systems may communicate with each other across the network. One end of the model deals with physical, electrical and mechanical interfaces, while the other end concerns services provided to applications. Each succeeding layer builds upon the services provided to it by the layer below and upwardly offers an enhanced service. The Transport resides in the middle of the architecture, at layer four.

As we have seen, a layer exists where cooperation between the distributed peer entities is governed by an application level protocol (Higher level protocol). Therefore a simpler basic hierarchical model involving only three layers can be used to study the Transport protocol, as shown in Figure 2-1.

In the top layer are located the processes which will cooperate through a high level protocol. If all these processes were in the same machine, their cooperation would take place through the local interprocess communication facility. When the processes are distributed in the network, the transport layer will provide a transport service, i.e. a distributed interprocess communication facility.

The Transport layer consists of distributed stations whose cooperation is governed by a Transport protocol. A station is in general, located at the respective end of the connection, close to the application users. The process A_i and the station A forms the "host" A and the Transport protocol is also known as a host-to-host protocol.

The transmission between stations takes place through the

service offered by the Transmission layer, which is the bottom layer of our basic hierarchy. We could also call it the Network Service, to be consistent with the OSI Reference Model.

2.2 PURPOSE OF A TRANSPORT PROTOCOL

This layer exists to provide transparent transfer of data between session entities. Transport protocols, usually, have end_to_end significance. An end can be thought as residing in the host installation site, or host nodes of a network. More references about the transport layer and end_to_end protocols could be found in [ISO 82], [ECMA 81], [NBS 83] and [OHBA 78]). The OSI proposal defines three phases of operation within the transport layer:

2.2.1. ESTABLISHMENT PHASE

The objective is to establish connections between peer transport functions on behalf of services requested from higher layers. Connection service quality can be negotiated during this phase. Services provided include:

- Selection of network services as a result of function of parameters such as throughput, transit delay, set-up delay, etc.
- Management of transport connection to Network layer connections.
- Establishment of appropriate data unit size.
- Transport of data from higher layers.

2.2.2. DATA TRANSFER PHASE

Services provided include:

- Blocking, concatenation, segmentation, multiplexing of connections.
- Flow Control in an end_to_end sense.
- Error Detection for lost, damaged, duplicated or misordered data units.
- Error Recovery to address problems detected in this layer or signalled from lower layers.
- Transport of high priority data which flow outside normal flow control mechanisms.

2.2.3. TERMINATION PHASE

This service allows either end of the session to terminate the connection with notification to the other party. Services include:

- Notification of termination reason.
- Identification of connection terminated.

2.3 CLASSES AND OPTIONS

The functions of the Transport layer have been organized into classes and options. A class defines a set of functions and options define functions which may or may not be used within a class.

According to the ISO reference model, five classes of protocol have been identified:

- a) CLASS 0: SIMPLE CLASS; the simplest class, for use with an elaborate network service such as X.25, which has a very low rate of errors.
- b) CLASS 1: BASIC ERROR RECOVERY CLASS; Can recover from network signalled errors (network disconnect or reset).
- c) CLASS 2: MULTIPLEXING CLASS; provides a way to multiplex several transport connections onto a single network connection. This class is further divide into subclasses depending on the use of flow control.
- d) CLASS 3: ERROR RECOVERY AND MULTIPLEXING CLASS; provides the characteristics of class 2 plus the ability to recover after a failure signalled by the network layer without involving the user of the transport service.

- e) CLASS 4: ERROR DETECTION AND RECOVERY CLASS; provides the characteristics of class 3, plus the detection of errors which occur as a result of the low grade of service available from the network server. The kinds of errors to be detected include: TPDU loss, TPDU delivery out of sequence, TPDU duplication and TPDU corruption. These errors may affect control TPDUs as well as data TPDU.

2.4 SERVICES ASSUMED FROM THE NETWORK LAYER

The network service has been identified in [ISO 82] as the provider of exchanging packets facilities to the transport entities. The logical characteristics of the network service specification are very similar to those of the transport Service. The network service, however, has certain additional elements, resets and clears, which requires special explanation. The reset service may be used by the network layer to report to its user the loss of data unrecoverable within the network service. The clear service is used by the network service to release an established network connection, usually as result of a user request to terminate the established connection or as indication of the inability to establish a requested connection.

The target of the transport protocol (and the mechanisms associated with it) is to overcome the network service weaknesses in order to achieve the service requested by the transport user. A list classifying network services in terms of the quality with respect to error behavior in relation to user requirements is presented here. The main purpose of this classification is to provide a basis for the decision regarding which class of transport protocol should be used on top of a given network connection:

- a) TYPE A. Network connection with acceptable residual error rate (for example not signalled by 'clear' or 'reset') and acceptable rate of signalled failures. For instance X.25. A transport protocol class 0 or 2 should be used.
- b) TYPE B. Network connections with acceptable residual error rate (for example not signalled by 'clear' or 'reset') but unacceptable rate of signalled failures. Transport protocol classes 1 and 3 has been designed to be us with this type of network connections.
- c) TYPE C. Network connections with residual error rate not acceptable to the network service user. Transport protocol class 4 has been designed to be used with this type of network connections. Packet radio networks are example of this type.

FIGURE 2-1. Three Layers Architecture Model.

CHAPTER 3

TRANSPORT PROTOCOL REQUIREMENTS

3.0 INTRODUCTION

The term "protocol requirements" addresses what the protocol is to do, i.e. the requisites of the data transfer phase of the transport protocol, in this case. In this chapter the transport protocol requirements are presented, in terms of:

- 1) Services and primitives of the Transport layer.
- 2) Services and primitives of the next lower layer which are used by the Transport protocol.
- 3) The internal functions and protocol elements.
- 4) The system transformations from its inputs into its outputs.

Since (1) and (2) are clearly specified in [ISO 82] and [ISO 83] respectively we do not discuss them here, but a summary of them is given in Appendix B for completeness.

In the following section the protocol functions are identified. In Section 3.2 the system transformations are presented in a graphical specification based on Finite State Machines (FSM).

3.1 INTERNAL FUNCTIONS REQUIRED:

The main data transfer functions of the Transport layer are:

- Receiving messages from the Transport user.
- Delivering messages to the Transport user.
- Message Packetizing/Depacketizing.
- Buffering messages/packets.
- Connections Multiplexing/Demultiplexing.
- Transmission of packets to the transmission layer.
- Reception of packets from the transmission layer.
- (Flow Control).
- (End_to_End error control).
- (Error corrections).

(Optional for Transport protocol class 2, but mandatory for higher classes).

[Used for Transport protocol class 3 and class 4].

(Used only for Transport protocol class 4).

The functions of the protocol can be organized as two logical pipelines (one for sending and one for receiving) as shown in Figure 3-2, with a cross-connection for handling buffers and for acknowledgment. Each pipeline is divided into three units. The highest one in the figure deals with the interfacing matters of the user protocol. And the lowest unit deals with the next lower layer interface. Of these two units the highest layer interface, is expected to be more complex due to the protocol service alternatives that it may provide. The lower layer interface only has to pass its information and requirements according to the next layer interface specification.

The middle unit of the pipeline contains the actual protocol data structures, like queues, time parameters, sequence numbers, retransmission queues, state variables. These data structures are affected not only by the previous two defined units, but by other system tasks like the time controller, the garbage collector (that handles the buffer cycles) and the protocol manager. The manager task is an aggregated approximate representation used here for the logic needed to correctly manage operations of connection, re-connection and disconnection.

3.2 TRANSPORT LAYER'S DATA TRANSFER PHASE TRANSITIONS USING FINITE STATE MACHINES (FSM):

To discuss performance we will consider the important sequences of operations of the protocol during the data transfer phase. In this section a graphical description of the protocol operations is presented based on Finite State Machine concepts. Discussions of this technique applied to communication protocols are found in [BOCHMA83a], [PIATKO79] and [PIATKO82]. The sequence of operations is represented via the sequence of states and the arrows represent actions executed by code which consume time, as studied in the next chapter.

3.2.1 CONNECTION DATA TRANSFER:

The class 4 data transmission FSM shown in Figure 3-3 has seven possible states. These seven states have been categorized here in four groups according to the transport protocol class 4 requirements. The four groups are: basic transmission states, flow control states, error detection states and error recovery

states. The frequency that each state is visited can be derived from analysis and/or assumptions made about these groups. In the basic transmission group two states are possible: one in which there is no data to send and so the FSM is waiting for the TS_user to submit some data, the other in which there is data to send and interactions with the network service entity take place. The use of flow control at the transport level is reflected in the state "waiting for credit". Error detection is represented by three states: a "waiting for ack" state which is needed because no new user data is accepted for transmission until all the data in the current T_DATA request has been acknowledged, a "network error recovery" state used in the recovery from network reset conditions, and a "waiting for peer" state which is entered upon detection of a Network reset condition. The error recovery capability provided by this class is introduced by a "retransmit data" state which is entered upon reception of a RJ TPDU, and the addition of a retransmit timeout event.

The data reception FSM for class 4 is shown in Figure 3-4, organized similarly. The basic reception state is the "waiting for peer" state, where the FSM remains until sufficient data TPDUs to fill a TSDU are received. The flow control state is the "waiting for TS_user" state, which is entered if no more credit is available. The error detection is represented by the "verify data sequence" state, where the validity of received data TPDUs is verified. Error recovery requires the additional event "inactive connection timeout".

3.2.2 EXPEDITED DATA TRANSFER

The expedited data transmit and receive FSMs are shown in Figure 3-5. Because only one expedited message can be sent at a time, the transmit FSM must wait for acknowledgement of each transmitted message before accepting a new message. Similarly, on the receive side, a received expedited message cannot be acknowledged until the TS_user has announced via a T_EX_DATA_ENABLE request that it is ready to accept a new message.

The error detection capabilities of the protocol require a network reset indication. In order to know when to retransmit an expedited data message, the error recovery needs a "retransmit timeout" as a possible event in the transmit FSM. This new event adds the possibility of out-of-sequence EA_TPDUs being discarded on the receive FSM.

3.2.3 DISCUSSION:

It is interesting to notice, through the representation of the data transfer phase of the Transport protocol in terms of FSMs, the persistence of short cycles or action paths which loop back to a home state. We can imagine them as single step actions. Most events which drive the protocol from one state to another and the actions performed in the course of such transitions have been service primitives or reception of TPDUs. Others, such as "window open", reflect changes in the values of local variables.

The only actions described so far are those that involve service primitives or the sending of TPDUs. In this way, it has been possible to concentrate on the functionality and interactions of the layer as a whole, not only with reference to the neighboring layers in the local machine, but with the peer entities at the remote site. The next step in the process of step-wise refinement is to describe the local environment within the layer.

FIGURE 3-1. GRAPHS NOTATION

Callable procedure or Monolithic Module: Can not be called simultaneously by several components (similar to Brinch Hansen "class" component). Every user has its own module instance.

Critical or Intercommunication Area: Defines shared data structures and all the operations that processes can perform on it, including the serialization operations to lock the service and admit only one (similar to concurrent Pascal Monitors).

Process: A self-contained software module made of private data structures and a sequential program that operates on the data, and running on a single processor element.

Packet Buffer: Area to store the actual data to be transmitted.

Control Flow,

Data Flow.

FIGURE 3-2. Functional Components of an ISO Transport Protocol.

- Basic Transmission States
- Flow Control States
- Error Detection States.
- Error Recovery States.

FIGURE 3-3. Class 4 Transport Data Transmission.

FIGURE 3-4. Class 4 Transport Data Reception.

FIGURE 3-5. Transport Class 4 Expedited Data Transfer

CHAPTER 4

DESIGN DECISIONS AND PARAMETERS

4.0 INTRODUCTION:

The implementation environment, interpreted widely to include all aspects of traffic demands, the network services, the computer hardware and operating system, has a deep and complex effect on design decisions. For this reason environmental issues and design issues will be discussed together. The body of the chapter covers a broad range of environment/design issues but concludes with a short list of particular alternatives and parameters for study in the next chapter.

In describing the system environment, it is necessary to take into account that:

- The description of the environment depends on the nature of the system actions and subsystem components assumed.
- The nature of the environment assumed is an integral part of the system description.
- Given that the environment itself, generally, is not under control of the system designer, the definition and relationship of the environment with the subsystems in the model is a major activity.

4.1 SET OF SYSTEM DESIGNS :

System design addresses "what the system (protocol) is" as a set of interacting parts with a common goal. The decomposition, allocation and organization of the protocol internal components must be analysed in relation with their respective performance characteristics, functional trade-offs and environment impact.

Examples of important design and operational environment issues and their characteristics are listed below and presented in more details in the following section.

- Operational Environment:

- Computer System Configuration (includes hardware, O.S. and peripherals available to the Transport protocol).
- Traffic Pattern (characteristics like average number of connections, average time per connection, traffic balance).
- Network Services Available (network reliability, availability of services).
- Availability of Special Hardware (like front-end processors).

- One End Protocol Interfaces (user interface, network service interface and system interface).

- Multiplicity of Connections (multiple transport connections which affects the organization of data structures).
- Matching End_to_End Paths Vs. Multiplexing.
- Separation of Error Recovery and Multiplexing.
- Buffer Management (storing outgoing and incoming messages, degree of buffer sharing among transport connections, buffering sharing among incoming and outgoing packets).
- Optional Features:
 - Measurement & Reporting (expense and complexity of the design Vs. usefulness to monitor subsystems).
 - Priority Traffic (like voice transmission).
 - Security (format, semantic, security policies).

4.2 OPERATIONAL ENVIRONMENT DESIGN ISSUES:

It is necessary that the technical requirements specify the operational environment in which the protocol will operate, as shown in the previous chapter. Environmental issues examined here include: computer system configuration, traffic pattern, network services available and special hardware available.

4.2.1 COMPUTER SYSTEM CONFIGURATION

The first aspect to be specified is the computer system configuration, this includes the hardware, operating system and peripherals available to the transport protocol. In this way it is possible to optimize design alternatives like the organization of protocol functional-units and the integration of the protocol to the system.

4.2.1.1 ORGANIZATION AND SYNCHRONIZATION OF PROTOCOL

FUNCTIONAL-UNITS:

A functional-unit is a sequence of instructions which perform a protocol function such as encoding, packetizing, checksum, etc. Since sequential functional-units must cooperate with each other, they must be able to communicate and synchronize their activities. Thus the machine architecture and/or programming environment, upon which processing units and synchronization facilities are built, become very important in the design. For instance, implementing every functional-unit of the protocol as a process could become very expensive in terms of network performance. This extra cost is due to the amount of overhead required, such as control switching, message passing, stack space. To minimize this overhead, functional-units can be grouped and implemented as callable procedures or monolithic units.

Let us look closely at some implementation alternatives for these functional-units and their synchronization effects (see Figure 4-1):

- 1) Callable Procedure, such as Concurrent Pascal "Class" processing unit.
- 2) Monitors, Brinch Hansen's Concurrent Pascal monitors for instance.
- 3) Ada tasks.

1. CALLABLE PROCEDURE:

A callable procedure is a processing unit that cannot be called simultaneously by several other units. It is passive until one of its entry routines is called. The executable instructions and execution time required to transfer control to a callable procedure is shown in Table 4-1, using the Basic PDP 11/60 Instruction Set. Since this unit cannot be shared by other processes the execution time is deterministic and there is no waiting involved.

TABLE 4-1: TYPICAL CALLABLE PROCEDURE SYNCHRONIZATION OPERATIONS

EXECUTER	DELAY ALLOW	I N S T R U C T I O N		R E Q U I R E D		TIME	MEMORY
		DESCRIPTION	HIGH LEVEL	LOWER LEVEL		(*)	CYCLES
CALLER	N	jump to routine	FUNC(PARAMS)	MOV #PAR,-(6)	1.20		2
				JSR #7,FUNC	2.50		1
RECEIVER	N	return	RETURN	RTS	1.50		2
				MOV (6)+,#PAR	1.02		2
TOTAL						6.22	7

* Time is given in microseconds, using the standard time for a PDP 11/60.

2. MONITORS:

Monitors as described in [HOARE 74], present a means of synchronizing concurrent processes and transmitting data between them. A monitor ensures that only one process at a time has access to the shared variables. The synchronization operations of a monitor are called "monitor procedures" (hence, the intercommunication using this facilities are referred to as "Procedure-Oriented Intercommunication").

Typical Procedure-oriented synchronization operations were coded using the basic set instructions of the PDP 11/60, to estimate the execution load requirements for process intercommunication. Table 4-2 summarizes these results (the low level code and instruction timing derivations are included in Appendix C). It is essential that only one program at a time actually succeed in entering a monitor procedure, and any subsequent call must be held until the previous call has been completed. To establish communication between two processors, the caller must acquire the monitor, deliver the message and leave the monitor, then the receiver enters the monitor and receives the message. The caller must execute the ENTER procedure, WRITE operation (representing the actual data deliver) and the LEAVE procedure. According to table 4-2, these operations take 36.08 microseconds in the best case, that is to say without having to wait to acquire the monitor. Otherwise it requires some extra processing time (independent of the time spend waiting until the monitor is liberated) for queueing the caller, and later for dequeuing and dispatching

the caller. Since the receiver performs the same operations as the caller to receive (READ) the transferred message, a single communication takes at least 72.16 microseconds. This processing time is much greater than the 6.22 microseconds required for a callable procedure to establish communication.

If the receiver accesses the monitor before the caller, there is some processing time involved in putting it self to SLEEP, and later on to become AWAKE (these procedures are included in Table 4-2 as optional procedures since their execution is probabilistic but not required). The increment in the number of processes communicating through the monitor, increases the amount of complexity in the control and size of this unit. The intercommunication delay also increases, due to the waiting time required to gain exclusive access to the monitor.

TABLE 4-2: TYPICAL PROCEDURE-ORIENTED SYNCHRONIZATION OPERATIONS

EXECUTER	DELAY ALLOW	I N S T R U C T I O N S		MINIMUM		MAXIMUM	
		DESCRIPTION	HIGH LEVEL	TIME (*) CYCLES	MEMORY (*) CYCLES	TIME (*) CYCLES	MEMORY (*) CYCLES
CALLER	Y	start sync. (ENTER)	WAIT(GATE)	15.58	19	38.89	58
	N	send signal (WRITE)					
	N	end of sync. (LEAVE)	IF HIPRI>0 THEN BEGIN HIPRI:=HIPRI-1 SIGNAL(PRIOR); END ELSE SIGNAL(GATE);	20.50	28	52.47	66
[CALLER	N	If the signal (AWAKEN) receiver is waiting for the caller to arrive, it must be awoken by the caller.	BEGIN IF EVENWAIT>0 THEN BEGIN EVEWAIT:=EVEWAIT-1; CTRLINK.HIPRIWAIT:= CTRLINK.HIPRIWAIT+1; SIGNAL(EVENTSEM); END; END	31.41	48	53.05	74
RECEIVER	Y	The receiver. (SLEEP) find no signal as it arrives, so it puts itself to sleep.	BEGIN EVEWAIT:=EVEWAIT+1; LEAVE(CTRLINK); WAIT(EVENTSEM); WAIT(CTRLINK.PRIOR); END	82.21	88	118.22	151
RECEIVER	Y	start sync. (ENTER)	WAIT(GATE)	15.58	19	38.89	58
	N	send signal (WRITE)					
	N	end of sync. (LEAVE)	IF HIPRI>0 THEN BEGIN HIPRI:=HIPRI-1; SIGNAL(PRIOR); END ELSE SIGNAL(GATE);	20.50	28	52.47	66

* Time is given in microseconds, using the standard time for a PDP 11/60 for the derivation of these times see appendix C.

[] Optional functions, but when executed both functions must be executed.

3. ADA TASK:

The idea of the ADA language, as proposed in [ICHBIA78], combines the idea of entry procedures, as used in monitors, and the idea of communicating to a process, rather than simply synchronization of the callers' use of the entry procedures. The ACCEPT statement allows the process (in ADA a task) to control the execution of the entry procedures. Only when one process has called an entry procedure and the process containing this entry procedure has executed an ACCEPT can the entry procedure be activated. This is known as a rendezvous.

There is a direct relation between the processes (or ADA tasks) and the monitors. For instance in the ADA style of resource manager, the branches of the SELECT statement correspond to the ENTRY procedure declarations in the other. The code representing the main loop of the process, SELECT, and ACCEPT statements performs exactly the same function as the mutual exclusion lock on the monitor, to serialize the request for service and admit only one.

Thus the algorithms of alternatives 2 and 3 will compute at roughly the same speed, and the same amount of information will be stored in each data structure. The same amount of user code will be executed in both systems. The same number of additions, multiplications, comparisons, and string operations will be performed. Therefore if basic processor characteristics are unchanged, then these will take precisely the same amount of computing power, and this component of the system performance will remain unchanged. Thus the basis for the decision between alternative 2 and 3 depends more on the environment.

CONCLUSION:

We can conclude that with reference to the organization and synchronization of entities in a protocol, implementing every functional-unit as a process will be too expensive, because of the overhead required. To minimize this overhead, the functions can be grouped or implemented as callable procedures. There are significant gains to be made by using this monolithic technique, which gives a better network performance while preserving the modular construction technique. On the other hand, it must be recognized when the application of this technique is not desirable. For instance, if the callable procedure handles the traffic coming from different layers or connections, it may be waiting (say, for an acknowledgement) on one call when another call is ready for processing; throughput would be reduced by this unnecessary delay. The functional-units organization, the number of concurrent processes and performance degradation can be estimated if the environment is known.

4.2.1.2 INTEGRATING THE PROTOCOL TO THE SYSTEM:

A decision should be made whether the transport protocol should operate in the user program areas or in the system program (i.e. kernel) area. The first is a straight-forward approach and could be done without the need to be an expert in the structure of the kernel. Unfortunately factors like process scheduling or operating system priority mechanisms could become a significant source of delay, and they need to be investigated.

On the other hand implementation in the kernel presumably alleviates much of the process scheduling. The design of applications which use the network could be facilitated. However, kernel implementation introduces new problems like: the need to have a person who is well-versed in the details of the host operating system, the well known and unresolved problem concerning the amount of kernel space desired and available, a decrease in the protocol portability as illustrated in [GRIFF184] and many new "head-aches" with the system maintenance.

4.2.2 TRAFFIC PATTERN:

Secondly it is necessary to define the particular traffic characteristics of the environment, such as the number of simultaneous connections that must be supported and the associated throughput requirements, the average time per connection, and traffic balance (file transfer vs. call-response traffic).

Examples of adapting the protocol design to match the traffic can be found in [CERF 78], [McQUIL78], and [MANNIN78]. They show that the systems and performance requirements, for environments where the traffic profile involves short transactions with long interarrival time, are best supported with nonconnection oriented datagram type protocols.

Also, in such environments, it is wise to provide schemes of sharing messages between several users which are communicating between the same systems, at some level. If these combining schemes are cleverly implemented it will improve transmission efficiency. The number of lower level messages are reduced while

preserving the requirement not to exceed certain delay constraints. One of the main characteristics in favor of short messages is not having to wait until long messages are filled.

4.2.3 NETWORK SERVICE AVAILABLE:

We should keep in mind that a transport protocol is usually only a part of an entire network architecture. In order to select an appropriate design and class of transport protocol to implement, it is necessary to consider the characteristics of the network service used and compare them against the user requirements. Some of the relationships between the transport and network service parameters have been identified in [RAGHUN83] for example. Some of the characteristics that need to be considered are described below.

4.2.3.1 NETWORK RELIABILITY:

Network reliability refers to the network service rate of loss of TPDU's, duplication, delivery out of sequence or errors. For instance, the class 4 transport protocol requires to send a checksum in every request. But it is possible to disable the checksum during data transfer, and save overhead. The decision to disable checksums must be made by considering the network error probability.

4.2.3.2 AVAILABILITY OF NETWORK SERVICES:

In designing the transport protocol, it is necessary to know the exact primitives and features that the network services offer to integrate the final product.

If both protocols, transport and network service, send acknowledgments for data, then the lower protocol could acknowledge the higher protocol's data and reduce the redundant control traffic. However this might be in violation of OSI principles. If both protocols retransmit lost data, then both will retransmit the same data to the destination along with duplicate buffering at the source. If both protocols operate end to end flow control, for instance using X.25 network service in a local area network environment, then one level may open its window as the other closes its window. This leads to conflicts and wide delay variations. If the protocols maintain data integrity on widely different data quantities, then one end is usually retransmitting while the other is waiting. This leads to suboptimal transfer throughput.

It is important to emphasize functions which the lower protocol does not perform. For example, if the network service does not support multiple connections, we will have to implement a multiplexing of transport connections in our transport layer if we require multiple connections in our system.

OBSERVATIONS:

Some general principles that must be kept in mind while designing the transport layer, taking into account the particular network services available, are:

- Minimize duplicate functions.
- Emphasize functions which the network service can not perform.

- Keep data integrity on similar size units of data as much as possible.
- Keep the lower protocols as independent as possible, to simplify evolutionary changes of any level. By allowing special levels to be substituted or used in parallel with the standard protocols, both the complete generality at each layer and the advantages of special protocols for special functions (like file transfer protocols) are accomplished.

4.2.4 AVAILABILITY OF SPECIAL HARDWARE:

Implementing the protocol on a dedicated front-end processor isolates the protocol entity from the kernel, simplifies implementation and maintenance. At the same time, it frees the host from unnecessary consumption of resources. In terms of protocol generality and efficiency, the management of the communication network can be improved. In [POH 79] for instance, the performance of two different implementations using front-end processors is studied. In [ANDLER81], they discuss the benefits of having a specialized microprocessor based communication device that performs many functions usually located in communication subsystem software (e.g. to reduce message passing overhead):

A simple example of an implementation for high throughput could be useful to understand the benefit of a front-end processor. Let us assume that a transport connection has been established with a fixed TPDU size of 256 bytes, ten of

which are used as DT_TPDU header. This size will permit 246 octets (or bytes) of data in each DT_TPDU. In order to accomplish a useful throughput of 1,000 bytes per second, a DT_TPDU must leave the host approximately every 250 milliseconds, or about 4 DT_TPDU per second. This is not an unreasonable goal. Yet to achieve 10,000 bytes per second throughput, a DT_TPDU must leave the host every 25 milliseconds. And to achieve 100 kilobytes per second throughput (not an unrealistic goal for file transfer on a Local Area Network, for instance), the DT_TPDU must leave the host every 2.5 milliseconds. This is still slow compared to the interrupt response time of most microprocessors available today, however if the protocol is implemented as a user-mode process, it may have to compete for CPU resources with other system processes, it may be necessary to wait a significant amount of time before it is scheduled to run. In many operating systems the environment switching requires 50 milliseconds or more. It is clear that the protocol in this operational environment needs to be integrated into a dedicated front-end processor.

The disadvantages associated with the front-end processor include: the cost of the additional computer to be dedicated to communications, and problems in its attachment to the host.

4.3 ONE END PROTOCOL INTERFACES DESIGN ISSUES:

There exist three interfaces to the transport protocol at each end of the connection: the user interface, the network interface and the system interface. The user interface provides the protocol service alternatives. The network interface has its own primitives and ways of passing and requesting available services. The system interface represent the link to system controller tasks like: timer, manager task, and possibly a garbage collector. The design issues for these interfaces concerns the generality of the services defined at each one. These issues are not examined further below.

4.4 MULTIPLICITY OF CONNECTIONS DESIGN ISSUES:

Many problems arise when one protocol implementation handles several connections. (See for example [COHEN 79] for a discussion of protocol multiplexing).

4.4.1 MATCHING END_TO_END PATHS VS. MULTIPLEXING:

Connection binding recognition has an important performance effect, especially in the Data Transfer phase. Two methods for binding connections are:

- Matching an End_to_End Path to a logical link, as shown in figure 4-2. This method is motivated by the need (or desire) to offer different grade of services and options to transport users, to simplify accounting procedures, and to improve efficiency through abbreviated addressing. Also it is appropriate if the users need to communicate at a fairly constant rate for a long period of time.

- Multiplexing multiple paths in the transport protocol to a single network connection, is shown in figure 4-3. Multiplexing has the advantage of reducing the establishment time and overhead of the network service. It is appropriate for low-duty-cycle users, if the alternative is to set up a network connection but not use it very much.

It is clear that both methods have their own application domains according to the traffic pattern of the implementation. Parameters such as average number of connections, average time per connection, traffic balance and, again, processing unit facilities available are needed to make a final design decision.

4.4.2 SEPARATION OF ERROR RECOVERY AND MULTIPLEXING:

Error recovery in the transport layer may be achieved in two ways: as a low-level function operating on the NSDUs; or as a higher-level function operating on the TPDUs. The first approach separates error recovery and multiplexing, based on the philosophy that the errors, being recovered from, are network errors which affect all packets. We will call this method "Bottom level recovery" since all the reliable delivery tools of the transport layer are implemented at the bottom and only once. By increasing the processing load at the bottom of the transport, there are gains not only in terms of simplifying evolutionary changes, and possibly reducing protocol size, but also in reducing the number of costly interactions with the connection entities. (See figures 4-4 and 4-5).

The second method, is called "demultiplexed" (shown in figures 4-6 and 4-7). This method performs error recovery after a received packet has been demultiplexed into its constituent TPDU's and routed to their particular connections. Thus, if many connections in the transport layer are being multiplexed on a network connection, then each transport connection must perform its own error recovery. The attitude reflected in this method is that each connection is responsible for implementing the degree of reliability required for its application. This, however, is straight-forward in that the principal mechanisms (sequence number and explicit acknowledgements) are available for flow control purposes. An error recovery capability consists simply of the addition of a retransmission queue for DT_TPDUs.

4.5 BUFFER MANAGEMENT

This design issue is a "local" implementation issue. It has effects only known at one-end of the network connection, but its performance impacts the total system, as will be shown in the last chapter. Topics in this area include:

- Preallocation of buffer space at the destination, or the decision to transmit copies of the data until acknowledged by the destination.
- Degree of buffer sharing among different connections.
- Buffering sharing among outgoing and incoming packets.

There is a growing literature on protocol performance analysing the effect on buffer management. For example, Yum [YUM 82] surveys the five "most popular" algorithms for buffer allocation and their effect on delay. From the execution workload point of view, it is necessary to investigate other effects of buffer management like data copying, buffer assignment and buffer fragmentation, which directly affect the processing load. Some implementation alternatives identified are explained below and their performance effects will be shown in chapter 5.

IMPLEMENTATION ALTERNATIVES:

Four implementation alternatives are shown graphically in figure 4-8. The difference between them are:

- 1) Physically copying the message and headers at each layer. Every layer has its own buffer space.
- 2) The message is stored in a buffer common to all layers. The headers and the pointer to the physical message are passed between layers and kept in their own header buffer space.
- 3) The message is stored in a buffer of predetermined fixed size, which is common to several layers. The highest layer of these layers is in charge of copying the message into the buffer, and must do any segmentation (fragmentation). Other layers only write to the fixed header area.

4) The message is again stored in a buffer common to all layers (as in (2)). The address to the message is passed to the other layers. The headers of the fragmented packets are kept in a common "headers buffer" space. This headers buffer, besides keeping all the lower layers' header fields, requires a pointer to the physical message. The address to the header buffer is passed among lower layers. Fragmentation may occur at any layer (only one is considered here); extra headers must be created and linked at fragmentation.

The advantages and disadvantages of these alternatives could be summarized as follows:

IMPLEMENTATION ALTERNATIVE	ADVANTAGES	DISADVANTAGES
Method 1	- Very flexible (variable packet and header size).	- Too much data copying.
Method 2	- The same as 1, but the copying time is reduced (only headers are passed at each layer).	- An extra buffer assignment (header buffer) is introduced.
Method 3	- No copying is required (except at the highest layer, but with a possible extra fragmentation time). - No buffer assignment (except at the highest layer).	- Predetermined packet size is required. - Fragmentation is allowed only in one layer. - Packets must be copied in special format.
Method 4	- Same as 3, but the special packet format is not required.	- An extra buffer assignment (header buffer).

The decision between these four alternatives must be on the basis of flexibility versus implementation complexity. This tradeoff will be discussed in Chapter 5.

4.6 OTHER ISSUES:

There are further design issues related with optional features of the transport protocol that we will not consider here, such as: —

- Measurement and Reporting,
- Priority,
- Security.

(a) Callable Procedures.

(b) Monitors: Procedure-Oriented Intercommunication.

(c) ADA Tasks.

FIGURE 4-1. Functional Units Organization Alternatives.

FIGURE 4-2. Connection Binding:
Matching End-to-End Paths.

FIGURE 4-3. Connection Binding:
Multiplexing.

FIGURE 4-4. Logical Components of the "Bottom Level Recovery" Method.

FIGURE 4-5. Queuing Model for the "Bottom Level Recovery" Method.

FIGURE 4-6. Logical Components of the "Demultiplexed" Method.

FIGURE 4-7. Queuing Model for the "Demultiplexed" Method.

(n) = Order of execution.

FIGURE 4-8(b). Method 2.

(n) = Order of execution.

FIGURE 4-8(a). Method 1.

FIGURE 4-8. Four Buffer Management Implementation Alternatives.

FIGURE 4-8(d). Method 4.

FIGURE 4-8(c). Method 1.

FIGURE 4-8. CONT.

CHAPTER 5

MEASUREMENTS ON TRANSPORT PROTOTYPE PROGRAMS

5.0 INTRODUCTION:

This chapter describes experiments on two prototype programs implemented to represent the data transfer phase of the transport protocol, one for sending and one for receiving data. The prototypes implement the four buffer alternatives in Chapter 4 with variations in the TSDU and TPDU size. Also a workload model to predict execution time/packet is constructed and verified using the results.

5.1 WORKLOAD COMPONENTS:

The execution time/packet can be classified into four phases as indicated in the execution model in Figure 5-1.

Phase 1 describes the transmission of a N+1 layer data unit from the input-buffer into the N-layer buffer. It is assumed that the messages (characters, bytes) that arrive at the interface are collected and placed at the N-layer_buffer as complete packets.

Phase 2 deals with the transmission of a N-level Data Unit from the N-layer buffer into the N-1 layer buffer. (This phase may also include the connection multiplexing/demultiplexing and binding processing).

Phase 3 describes the transmission of a received message from the N-1 layer buffer to the N-buffer (including the demultiplexing/multiplexing conversions and the additional processing overhead).

Phase 4 corresponds to the transmission of the received message to the N+1 layer.

Using the results of this chapter, the service time for each phase can be estimated. This model can then be used, as shown in Chapter 6, to study protocol implementation options, behavior, performance and side effects.

5.2 WORKLOAD ESTIMATION:

The workload estimation is done in two steps. The first step is to describe the protocol implementation as a set of components with predicted average execution demands. Figure 3-2 presents a structural and interface specification of the transport protocol. The processing model in Figure 5-2 shows the execution sequence and the points where processing may be suspended (queued) as a packet is processed for transmission.

The second step is to estimate the execution time for each module. Obviously this will depend on the language environment, compiled code, and other implementation details. However a ideal lower bound workload estimation can be derived for each module here. The main subsystems were coded in "C" for a VAX 750 system (running UNIX). The UNIX statistical facility GPROF was used for measurements.

5.2.1 ORGANIZATION OF THE PROTOTYPES

This section provides an overall view of the operation of the prototype software that was developed. It was organized as two independent processes. The first process describes the sending pipeline of one transport connection, the second one the receiving pipeline, as illustrated in Figure 3-2. During the experiment the 'send' and 'receive' processes were tested separately to avoid extra system-dependent factors, like interprocess communication, that were described previously, in Chapter 4. Each of the functional components within these pipelines was represented as a callable procedure. The transport connection is assumed to be initialized, since in this thesis we are interested in the Data Transfer Phase only. That means that the sender and receiver identification addresses are established, as well as the next output packet sequence number (NOPS) and the next input packet sequence number (NIPS). The implementation details of both processes, sender and receiver, are described below.

5.2.1.1 SENDER: As shown in Figure 5-3, the sender process is constructed from seven modules, including three modules that handle functions related to the buffer pool.

MAIN: This procedure accepts the following input values: TSDU size (MAXMSG), the size of the TPDUs (MAXPAC) and the number of TSDUs to send (NOPACKS). It calls the INIT procedure, and then loops calling SESSEND and TRANSEND until all messages (TSDUs) have been sent.

INIT: This module initializes those variables that must be initialized during the establishment phase of a transport connection.

SESEND (Session Send): This procedure simulates a Transport Service user. It places the message to be transmitted (TSDU) in a buffer, calling ALLOC, and then signals TRANSEND.

TRANSEND (Transport Send): The functions within this module represent different functions of an ISO transport layer. It was implemented in different forms in order to observe and measure the effects in execution performance of such changes. In general it receives a request from SESEND to send a TSDU, fragment it in several TPDUs if required and then cycles calling ENCODE and placing the TPDUs in the outgoing queue, using QUEUE, until the complete TSDU has been sent.

ENCODE: The parameters necessary to prepare an outgoing TPDU are here. They include: the NOSN, sender and receiver identification. If the connection was initialized to provide checksum over the TPDUs to be transferred, it calls CHECKSUM.

CHECKSUM: This procedure provides data assurance by the use of a CRC check which, when a verifying checksum is carried out after transmission, will give a zero value if no error occurred.

QUEUE: Simulates the queueing of the outgoing message to be picked up later by the Network Service. The TPDU is queued only if the WINDOW is open. The queue of outgoing messages is formed by a pointer to the TPDU, the NOSN, the length of the packet and a field, TIME, which will be filled when transmitted.

Upon transmission of the packet, the WINDOW is advanced and the TSDU buffer space is set free by procedure "LIBRE".

BUFFER POOL MODULES:

ALLOC, MORECORE and LIBRE, represent respectively: the general storage allocator, the operating system storage obtainer, and the storage releaser as described in [KERNIG78].

BALLOC and BLIBRE are the corresponding procedures in a fixed-buffer size format, as represented in Fig. 4-8(c). Although these routines do not require dynamic memory allocation (MORECORE), a buffer initialization (BUFFINIT) is required to link and create the fixed buffers. It also required other procedures, "INSERT" and "EXTRACT", to fill in or read the value of the buffer. A private instance of these procedures was implemented for each "layer" to avoid the misinterpretation of the profile output.

5.2.1.2 RECEIVER:

This process is similar to the sender process, as shown in Figure 5-4.

MAIN: This procedure accepts as input, the four values: NSDU size (MAXFRAME), the size of the TPDUs (MAXPAC), the number of TPDUs that constitute an TSDU (NOMESG) and the number of NSDUs to receive (NOPACKS). It cycles calling RECIBIR, and TRANSRECV until all the frames are received and delivered to the SESSRECV.

RECIBIR (Spanish for receive): This module initializes variables and generates a "good" incoming frame. It places the proper checksum value of the NSDU by calling CHECK1. CHECK1 is the same as CHECKSUM (renamed to avoid mixing times and statistical measurements). Then RECIBIR places the frame in a buffer, calling ALLOC.

TRANSRCV (Transport Receiver): The packet reception functions of the transport layer are carried out in this procedure. First, it calls DECODE. If the incoming packet is not a good one, TRANSRCV calls SENDING to send a reject packet (RJ_TPDU), using all the facilities described for the outgoing packets (i.e. using SENDING ENCODE and QUEUE). If the received packet is good and has a new acknowledgment, the window is advanced, the retransmission pointer is updated, and the packet buffer is freed (BLIBRE). If the incoming packet is a data transfer packet (DT_TPDU) and its sequence number is the same as the next input packet sequence number (SEQNO = NIPSN), TRANSRCV increments NIPSN and calls SENDING to send an acknowledgment (AK_TPDU). When end of TSDU is indicated it calls SESSRCV and delivers the entire message (TSDU).

SESSRCV (Session Receiver): This procedure simulates a Transport Service user. Its only function is to accept the TSDU. It is useful for statistical matters.

DECODE: This module extracts the header values of an incoming frame and sets flags such as BADCODE, BADCHECKSUM, EOMESG.

The other modules (ENCODE, CHECKSUM and QUEUE) are as described for the sender process.

5.2.2 EXPERIMENT DESIGN AND RESULTS

A number of issues need to be considered due to the large number of parameters involved, as mentioned in Chapter 4. To keep the analysis simple it was necessary to limit the number of experiments to a small set of parameter combinations. The data collected during the experiments were analysed for impact on the host. To obtain measurement values the GPROF facility of the UNIX operating system was used. GPROF is a tool to generate what in a UNIX system is called a profile of a program. A profile is a list of percentages of time spent executing each procedure. By examining the profile, time-critical or inefficient parts of the program can be identified for instance. In the case of GPROF, the first listing contains the total execution time and call counts for each of the functions of the program, sorted by decreasing time. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents. For details of how to use GPROF see for example [McGILT83]. The description and results of the evaluations are presented bellow.

5.2.2.1 OUTGOING TRAFFIC:

Two trials were carried out, using two different types of buffer management.

FIRST TRIAL:

In the first trial, the buffer management technique is as shown in Figure 4-8(a). The messages to be transmitted are passed and copied at each layer. Every layer has its own buffer space. Results for different packet sizes, and different levels of TSDU fragmentation, are summarized in Tables 5-1 and 5-2.

TABLE 5-1: PROTOCOL PROCESSING TIME (First Trial: sending)

Bytes per TSDU	Bytes per TPDU*	Total Useful Data (in bytes)	Elapsed Time (sec)	Rate of Useful Data (bytes/sec)	Rate of Total Data (bytes/sec)
119	128	1,190,000	85.90	13,853.32	14,901.05
247	256	2,470,000	160.02	15,435.67	15,998.00
503	512	5,030,000	314.75	15,980.88	16,266.88
1015	1024	10,150,000	612.74	16,564.94	16,711.82
2039	2048	20,390,000	1213.11	16,808.04	16,882.23
512	137	5,120,000	352.56	14,522.35	15,543.45
512	265	5,120,000	328.82	15,570.83	16,118.24
512	521	5,120,000	317.80	16,110.76	16,393.44
1000	1009	5,120,000	309.84	16,524.66	16,673.38
1000	109	5,120,000	364.81	14,034.70	15,297.83
500	509	5,120,000	321.83	15,909.02	16,195.38

* Including a 9-byte header.

TABLE 5-2: TIME SPENT PER PROTOCOL FUNCTION (First Trial: sending)

ROUTINES	TIME IN MICROSECONDS			
	Per TSDU byte	Per TPDU byte	Per TSDU	Per TPDU
Sessend	11.3			
Checksum		37.5		
Transend		11.0		
Libre			97	97
Alloc			90	90
Queue				85
Encode				117
TOTAL	11.3	48.5	187	389

FRAGMENTATION OVERHEAD:

In Table 5-1, we can observe the overhead generated by processing small packets versus longer packets, and the effect of packet fragmentation. For instance, row six of the Table 5-1 says that using 137 byte TPDUs takes 352.56 seconds to transfer 5,120,000 bytes of useful data. However, row eight says that the same amount of useful data (5,120,000 bytes) is transferred in 317.80 seconds using 521-byte TPDUs. From the strict point of view of the Transport processing time, it will be better to send large packets with no fragmentation. The extra work that must be done for each fragmented packet, assuming that only one acknowledgment packet per TSDU is sent back from the receiver to the source, is:

ROUTINE	TIME	COMMENTS
CHECKSUM	9 * 37.5	Checksum of the header bytes.
TRANSEND	9 * 11.0	Processing time of the header bytes at the transport level.
ALLOC	90	Alloc. of an extra packet storage.
LIBRE	97	Freeing the extra packet storage.
QUEUE	85	Queueing the extra outgoing packet.
ENCODE	117	Preparing the extra outgoing packet.
TOTAL	825.5	microseconds per extra packet.

A formula to calculate the fragmentation overhead of this buffer management technique, as implemented here, is given by:

$$F = (\text{No. of TPDU} - \text{No. of TSDU}) * 825.5 \text{ microseconds/TSDU (5-1)}$$

BYTE-TRANSFER OVERHEAD:

In Table 5-2 the total execution time of the protocol has been broken down according to the time spent in each procedure. For example, row two of Table 5-2 says that the routine CHECKSUM spends 37.5 microseconds for each byte sent, while row six says that routine QUEUE spends 85 microseconds for each outgoing packet. Obviously the procedures that work on a per-byte basis dominate the total execution time of the protocol, as can be clearly seen in Figures (5-6)-(5-7), (Execution Time/byte = 48.5 microsec). Leaving out SESSEND, because it is not part of the transport protocol, those procedures are CHECKSUM and TRANSEND. Any improvements to the execution of these procedures will result in improvement of the total execution time.

For instance, the procedure CHECKSUM was changed to place the checksum field at the end of the outgoing packet, instead of using the checksum field inside the header area. The average processing time of CHECKSUM came down from 37.5 microsec./byte (see Table 5-2, row 2) to 36.0 microsec./byte since the final "value according to internal position" calculation is avoided.

Another idea would be to use hardware to implement such functions. For instance the INTEL 8274, Multiple Protocol Serial Controller (MPSC), generates automatically a CRC-16, while transferring the packet. Although the maximum baud rate of the 8274 is 880 Kbaud, if we use the Intel MPSC and place the CRC value at the end of the packet, the execution time per TPDU of this CHECKSUM hardware implementation is given by:

$$\text{Exec. Time/TPDU} = \text{Transferring Time} + \text{Copying} + \text{Buffer Alloc.} \quad (5-2)$$

$$(\text{in microsec.}) = 9.2 \text{ per byte} + 11 \text{ per byte} + 187 \quad (5-3)$$

which means that for TPDUS longer than 10 bytes this CHECKSUM hardware implementation requires less time than the software routine.

TRANSEND is the other procedure whose execution time is proportional to the number of bytes it handles, because it copies data from the session buffer into the transport buffer. To see how a particular environment feature affects the overall system performance, TRANSEND was coded in two ways, using special register variables or normal variables, while keeping everything else identical. The register variables code reduces the average processing time of the routine from 14.2 microsec. to 11 microsec. per byte.

SECOND TRIAL:

The "per byte" execution time as reported from the first trial, appears to be important. To decrease processing load, a different buffer management technique was implemented so that packets are never copied. Each packet is always given one (or more, if needed) full packet buffer(s). This management technique is illustrated in Figure 4-8(b). The message to be transmitted is placed in a "common fixed_packet_size" buffer by routine INSERT. INSERT is called by the highest layer, the session layer in this case. All other layers write only in their "header space". The same test cases used during the first trial ~~were used for the second trial.~~ The results for these second trial are summarized in Tables 5-3 and 5-4 below.

TABLE 5-3: PROTOCOL PROCESSING TIME (Second Trial: sending)

Bytes per TSDU	Bytes per TPDU*	Total Useful Data (in bytes)	Elapsed Time (sec)	Rate of Useful Data (bytes/sec)	Rate of Total Data (bytes/sec)
119	128	1,190,000	70.40	16,903.41	18,181.82
247	256	2,470,000	131.10	18,840.48	19,527.08
503	512	5,030,000	263.60	19,081.94	19,423.37
1015	1024	10,150,000	506.61	20,035.14	20,212.79
2039	2048	20,390,000	990.04	20,595.13	20,686.03
512	137	5,120,000	281.47	18,190.22	19,469.22
512	265	5,120,000	268.14	19,094.50	19,765.79
512	521	5,120,000	259.44	19,734.81	20,081.72
1000	1009	5,120,000	251.12	20,388.66	20,572.16
1000	109	5,120,000	288.47	17,748.81	19,346.21
500	509	5,120,000	259.25	19,749.28	20,104.76

* Including a 9-byte header.

TABLE 5-4: TIME SPENT PER PROTOCOL FUNC. (Second Trial: sending)

ROUTINES	TIME IN MICROSECONDS			
	Per TSDU byte	Per TPDU byte	Per TSDU	Per TPDU
Insert (session)	10.8			
Checksum		37.5		
Sessend2			89	
Blibre			63	
Transend2			155	45
Balloc			122	27
Insert (transp.)				128
Encode				117
Queue				85
TOTAL	10.8	37.5	429	402

DISCUSSION:

By changing TRANSEND.(now TRANSEND2) to work with fixed size packets instead of on a "per byte" basis the total protocol execution time came down 22%. Compare results in Table 5-1 and Table 5-3. For instance, row 8 of Table 5-3 says that to transfer 5,120,000 bytes using 137-byte TPDUs in the second trial took 259.44 seconds, however in the first trial (see Table 5-1, row 8) this transfer was done in 317.80 seconds (the execution time was 22% higher than the second trial). The byte-transfer execution time is now 37.5 microsec.(48.5 microsec. in the previous trial).

The routine INSERT (session) does all the copying of the data into the buffer (this function was done by SESSEND before) at the same time it fragments the message if needed. INSERT when called by the transport, physically inserts the header of the packet in the packet buffer. BALLOC and BLIBRE substitute ALLOC and LIBRE, respectively, as explained before. The other routines were not changed. Notice that TRANSEND2 and BALLOC have fixed processing time per outgoing TSDU packet. However if fragmentation is required, an extra 45 and 27 microseconds per extra TPDU, respectively, are added directly to the processing time. The cost of fragmentation is the processing of:

ROUTINE	TIME	COMMENTS
CHECKSUM	9 * 37.5	Checksum of the header bytes.
TRANSEND2	45	Transport overhead due to repetitive functions applied per TPDU.
BALLOC	27	Buffer management overhead required for fragmentation.
INSERT	128	Insertion of a packet header.
QUEUE	85	Queueing the extra outgoing packet.
ENCODE	117	Preparing the extra outgoing packet.
TOTAL	739.5	microseconds per extra packet.

The fragmentation overhead of this case will be:

$$F = (\text{No. of TPDU} - \text{No. of TSDU}) * 739.5 \text{ microseconds/TSDU (5-4)}$$

5.2.2.2 INCOMING TRAFFIC:

The same two trials that were used for outgoing traffic, requiring different buffer management techniques, were applied to incoming traffic.

FIRST TRIAL:

The first trial uses the buffer management technique where the message is being copied at each layer. The results of this trial is summarized in Table 5-5.

TABLE 5-5: TIME SPENT PER PROTOCOL FUNCTION (First Trial: recv.)

ROUTINES	TIME IN MICROSECONDS			
	Per DT_TPDU byte	Per AK_TPDU byte	Per DT_TPDU	Per AK_TPDU
Checksum	36.0	37.5		
Transrecv	10.0			
Copy		9.2		
Decode			118	
Alloc			90	
Libre			97	
Encode				117
Sending (acknow.)				98
Queue				85
TOTAL	46.0	46.7	305	300

DISCUSSION:

The CHECKSUM has a lower processing time receiving a DT_TPDU (36 microsec/byte) than for sending an AK_TPDU byte (37.5 microsec/byte). In receiving a DT_TPDU, CHECKSUM only needs to find the CRC value, while in sending an acknowledgment, the CRC value must be inserted in the packet.

When an incoming packet is received, a new buffer is allocated (ALLOC) where the packet is transferred by TRANSRECV. However if a new incoming packet is detected an acknowledgment packet needs to be sent (SENDING). Instead of allocating a new buffer space for the AK_TPDU, it has been assumed here that a standard AK_TPDU already exists. To send an acknowledgment, requires copying (COPY) the particular information in the standard AK_TPDU buffer.

The overhead incurred to acknowledge a new incoming TPDU is presented in Table 5-5 (46.7 microsec/AK_TPDU + 300 microsec/AK_TPDU-BYTE). The ISO reference model does not use the piggybacking facility for level 4 acknowledgments, ACK packets are interleaved with data packets. However, if the acknowledgments are piggybacked to the outgoing traffic the following processing times are avoided:

CHECKSUM	37.5	microseconds/byte
ENCODE	109.0	microseconds/acknowledgment
QUEUE	85.0	microseconds/acknowledgment

TOTAL SAVED TIME: 37.5 AK_TPDU byte+ 199 AK_TPDU microsec. (5-5)

For instance, if the acknowledgments are 9-byte long the total saved time/ACK_TPDU is 446.5 microseconds. Notice that SENDING and COPY are still needed.

SECOND TRIAL:

For the single common buffer with a fixed packet size, the results are summarized in following Table:

TABLE 5-6: TIME SPENT PER PROTOCOL FUNCTION (Second Trial: recv.)

ROUTINES	TIME IN MICROSECONDS			
	Per DT_TPDU byte	Per AK_TPDU byte	Per DT_TPDU	Per AK_TPDU
Checksum	36.0	37.5		
Extract			143	
Transrecv2			187	
Decode			118	
Insert				128
Encode				117
Sending (acknow.)				98
Queue				85
TOTAL	36.0	37.5	448	428

EXTRACT and INSERT are overheads required due to the "fixed-packet-size" buffer. They eliminate of having to allocate and then liberate a private buffer, and free TRANSRECV2 from having to copy each message. Everything else was kept the same as the previous trial.

FIGURE 3-1. Four Phases Model of a Computer Communications Protocol.

FIGURE 5-2. Queuing model for the transport packet-execution.

FIGURE 5-3. SENDER Process Modular Representation.

FIGURE 5-4. RECEIVER Process Modular Representation.

FIGURE 5-5. Execution Time Distribution for Outgoing Traffic (TSDU size=128 octets).

FIGURE 5-6. Execution Time Distribution per Routine for sending 10,000 128_Byte_TSDUs.

FIGURE 5-7. Execution Time Distribution per Routine for sending a 2048_Byte_TSDU.

CHAPTER 6

PROTOCOL DESIGN CASES:

6.0 INTRODUCTION:

With the estimated execution-time parameters for the protocol components founded in the previous chapter, it is possible to predict the execution load of different protocol designs, although it is difficult to obtain process times before the actual code is written. The goal of this chapter is to look at some examples of how different implementation options can be analysed for their processing time requirements. In section 6.1 the four different buffer management techniques presented in chapter 4 are analysed. In section 6.2 the maximum throughput capacity for the two cases described in the previous chapter (sending and receiving using two different types of buffers) is analysed. Section 6.3 presents the transit time performance for the same two cases. Then, in section 6.4 a comparison between CLASS 4 and CLASS 2 Transport protocols, in terms of mean execution time per TSDU is derived.

6.1 BUFFER MANAGEMENT TECHNIQUES ANALYSIS

6.1.1 PROCESSING TIME FOR HANDLING OUTGOING MESSAGES:

In order to calculate the mean execution time for outgoing messages using the buffer management techniques presented in

Chapter 4, the functional requirements of each technique can be summarized by the calls shown below. The following assumptions are also made:

- Assumptions:
- (i) There are 4 layers.
 - (ii) Each layer has 10-byte header.
 - (iii) No fragmentation is allowed.
 - (iv) The final packet retrieval for transmission by the lowest level was not taken into account.
 - (v) Garbage collection processing and scheduling time was not taken into account.

PROCESSING TIME PER MESSAGE FOR EACH METHOD:

Method 1:

ROUTINE	FUNCTION	No. OF CALLS	TIME PER CALL	TOTAL PROCESSING TIME PER ROUTINE
ALLOC	Buffer allocation	4	90	360
LIBRE	Buffer liberation	4	97	388
COPY	Data copying	4	$9.2 \cdot B1$	$36.8 \cdot B1$
COPY	Header copying	(1)	$9.2 \cdot 100$	920
TOTAL PROCESSING TIME PER PACKET				$1668 + 36.8 \cdot B1$

Where:

B1 Is the number of bytes per message.

(1) The first layer only copies its 10 header octets, but the fourth layer will copy 40 header octets. The total header octets that are copied in processing four layers will be 100 octets.

PROTOCOL DESIGN CASES

6-3

Method 2:

ROUTINE	FUNCTION	No. OF CALLS	TIME PER CALL	TOTAL PROCESSING TIME PER ROUTINE
ALLOC	Main buffer alloc.	1	90	90
LIBRE	Main buffer liber.	1	97	97
COPY	Data copying	1	9.2*B1	9.2*B1
ALLOC	Headers buffer alloc.	3	90	270
LIBRE	Headers buffer liber.	3	97	291
COPY	Header copying	-	9.2*100	920

TOTAL PROCESSING TIME

1668 + 9.2*B1

Method 3:

ROUTINE	FUNCTION	No. OF CALLS	TIME PER CALL	TOTAL PROCESSING TIME PER ROUTINE
BALLOC	Main buffer alloc.	1	149	149
BLIBRE	Main buffer liber.	1	63	63
INSERT	Insertion of data.	1	10.8*B1	10.8*B1
INSERT	Insertion of headers	4	128	512

TOTAL PROCESSING TIME

724 + 10.8*B1

Method 4:

ROUTINE	FUNCTION	No. OF CALLS	TIME PER CALL	TOTAL PROCESSING TIME PER ROUTINE
ALLOC	Main buffer alloc.	1	90	90
LIBRE	Main buffer liber.	1	97	97
COPY	Data copying	1	9.2*B1	9.2*B1
ALLOC	Headers buffer alloc.	1	90	90
LIBRE	Headers buffer liber.	1	97	97
COPY	Header copying.	-	9.2*40	368

TOTAL PROCESSING TIME

742 + 9.2*B1

The processing time per outgoing message can be calculated from these values, as shown in the following table:

TABLE 6-1: BUFFER MANAGEMENT TECHNIQUES PROCESSING TIME
PER TSDU WITHOUT FRAGMENTATION (in microseconds).

B1 (Bytes per Message)	METHOD 1	METHOD 2	METHOD 3	METHOD 4
64	4023.20	2256.80	1415.20	1330.80
128	6378.40	2845.60	2106.40	1919.60
256	11088.80	4023.20	3488.80	3097.20
512	20509.60	6378.40	6253.60	5452.40
1024	39351.20	11088.80	11783.20	10162.80
2048	77034.40	20509.60	22842.40	19583.60

DISCUSSION:

As expected, Method 2 is always better than method 1. The main difference between methods 1 and 2, is that the second one avoids copying the message at each layer. For the case where very long packets are sent, method 2 is better than method 3. Method 3 has a higher execution time per byte transmitted, although its fixed time per packet transmitted is lower than method 2. But if the traffic pattern of our environment is that of great amount of short messages, Method 3 will be better. Method 4 seems to have the smallest overhead, since it is based on the best qualities of Method 2 and 3.

6.1.2 FRAGMENTATION OVERHEAD FOR EACH METHOD:

Similarly, other parameters such as B_1 , M , Number of layers, header size at each layer, can be change to estimate their workload effects. For instance, if we consider four layers, the header size to be 10 bytes per layer, but the B_1 byte-messages are assumed to be fragmented at the fourth layer into M TPDU's, the mean processing time per message can be calculated from the following equations:

METHOD PROCESSING TIME PER MESSAGE (In microseconds)	
Method 1:	$1389 + 36.8 B_1 + 279 M$
Method 2:	$1389 + 9.2 B_1 + 279 M$
Method 3:	$569 + 10.8 B_1 + 155 M$
Method 4:	$463 + 9.2 B_1 + 279 M$

where M is the average number of TPDU's per message.

DISCUSSION:

Some values calculated from these equations are given in Table 6-2. Notice that in all cases fragmentation causes some extra processing overhead (coefficients of M in the equations). However, less processing time is required to fragment a message into M TPDU's than to send M different messages. This fact is due to the processing time saved at the higher layers, before the fragmentation is done.

Method 3 requires more processing time for long messages due to its high execution time per byte transmitted. However it has the lowest fragmentation overhead among the four methods (155 microsec/TPDU). Methods 1 and 2 are heavily penalised by fragmentation (1389 microsec/TPDU), due to extra buffer acquisitions

and header copying requirements. The fragmentation overhead for method 4 (279 microsec.) is higher than for method 3, however its fixed processing time per byte and per message is lower.

Figure 6-1 compares the mean execution processing time versus different outgoing message sizes, using 4 layers and 2 TPDUs per message ($M=2$), as shown in Table 6-2. Notice how method 3 requires less processing time for small messages than the other methods, but longer processing time for long messages. Figure 6-2 shows the mean execution time required for each method to send a 1024 byte message using different numbers of packet fragmentations. If the traffic pattern of the particular implementation is known, an optimal buffer management technique can be chosen.

6.2 THROUGHPUT PERFORMANCE

The principal result that we are able to derive from our workload model is the mean execution time in level 4, i.e. Time T_s for passing a complete message to the Network Service, and time T_r to transfer a message from the Network Service to the Transport user connection.

$$T_s = \text{Mean Send Time} \quad (6-1)$$

$$\begin{aligned} &+ (\text{Mean Acknowledgment/Send}) (\text{Acknowledgment Time}) \\ &+ (\text{Mean Reject/Send}) (\text{Reject Time}) \\ &+ (\text{Mean Retransmission/Send}) (\text{Retransmission Time}) \end{aligned}$$

$$T_r = \text{Mean Receive Time} \quad (6-2)$$

$$\begin{aligned} &+ (\text{Mean Acknowledgment/Receive}) (\text{Acknowledgment Time}) \\ &+ (\text{Mean Reject/Receive}) (\text{Reject Time}) \end{aligned}$$

TABLE 6-2: BUFFER MANAGEMENT TECHNIQUES

PROCESSING TIME PER TSDU WITH FRAGMENTATION (in microseconds)

ASSUMPTIONS:

- i) No. of layers: 4 layers.
- ii) Header size (at each layer): 10 octets

M	BYTES PER MESSAGE	METHOD 1	METHOD 2	METHOD 3	METHOD 4
1	64	4023.20	2256.80	1415.20	1330.80
1	128	6378.40	2845.60	2106.40	1919.60
1	256	11088.80	4023.20	3488.80	3097.20
1	512	20509.60	6378.40	6253.60	5452.40
1	1024	39351.20	11088.80	11783.20	10162.80
1	2048	77034.40	20509.60	22842.40	19583.60
2	64	4302.20	2535.80	1570.20	1609.80
2	128	6657.40	3124.60	2261.40	2198.60
2	256	11367.80	4302.20	3643.80	3376.20
2	512	20788.60	6657.40	6408.60	5731.40
2	1024	39630.20	11367.80	11938.20	10441.80
2	2048	77313.40	20788.60	22997.40	19862.60
4	64	4860.20	3093.80	1880.20	2167.80
4	128	7215.40	3682.60	2571.40	2756.60
4	256	11925.80	4860.20	3953.80	3934.20
4	512	21346.60	7215.40	6718.60	6289.40
4	1024	40188.20	11925.80	12248.20	10999.80
4	2048	77871.40	21346.60	23307.40	20420.60
8	64	5976.20	4209.80	2500.20	3283.80
8	128	8331.40	4798.60	3191.40	3872.60
8	256	13041.80	5976.20	4573.80	5050.20
8	512	22462.60	8331.40	7338.60	7405.40
8	1024	41304.20	13041.80	12868.20	12115.80
8	2048	78987.40	22462.60	23927.40	21536.60
16	64	8208.20	6441.80	3740.20	5525.80
16	128	10563.40	7030.60	4431.40	6104.60
16	256	15273.80	8208.20	5813.80	7282.20
16	512	24694.60	10563.40	8578.60	9637.40
16	1024	43536.20	15273.80	14108.20	14347.80
16	2048	81219.40	24694.60	25167.40	23768.60
32	64	12672.20	10905.80	6220.20	9979.80
32	128	15027.40	11494.60	6911.40	10568.60
32	256	19737.80	12672.20	8293.80	11746.20
32	512	29158.60	15027.40	11058.60	14101.40
32	1024	48000.20	19737.80	16588.20	18811.80
32	2048	85683.40	29158.60	27647.40	28232.60
64	64	21600.20	19833.80	11180.20	18907.80
64	128	23955.40	20422.60	11871.40	19496.60
64	256	28665.80	21600.20	13253.80	20674.20
64	512	38086.60	23955.40	16018.60	23029.40
64	1024	56928.20	28665.80	21548.20	27739.80
64	2048	94611.40	38086.60	32607.40	37160.60

The system capacity will be influenced primarily by these two processing times and the network link capacity. Ideally, these three factors would contribute equally, so that the nodes and link loading would be evenly balanced, but in practice, it may well be that the overall system loading will be limited predominantly by one of them.

6.2.1 MEAN EXECUTION TIME AT LEVEL 4 FOR THE TWO TRIALS

FIRST TRIAL:

From the values for the first trial in Tables 5-2 and 5-5, and equations (6-1) and (6-2) we have that the mean execution time (in microseconds) at transmitter and receiver, per TSDU for the buffer management method 1 are:

$$T_s = M [48.50 B + 389 + AK ((46.0 + 9.2)H_1 + 118 + 97)] \quad (6-3)$$

and

$$T_r = M [46.0 B + 305 + AK ((37.5 + 9.2)H_1 + 300)] \quad (6-4)$$

Where:

B = Bytes per TPDU,

M = Average number of TPDUs per TSDU,

AK = Average number of acknowledges per TPDU,

H₁ = bytes per AK_TPDU.

Let us consider the effect of transmitting with errors.

If a packet is damaged the receiver generates a RJ_TPDU and sends

it back. Assume a packet is damaged with probability RJ, and that a RJ_TPDU consists of H2 bytes. The reception of a RJ_TPDU generates extra processing to the original sender. The processing effect of transmitting with errors adds to the mean execution time at the transmitter the term:

$$RJ ((46.0 + 9.2) H2 + 118) \quad (6-5)$$

And at the receiver:

$$RJ ((37.5 + 9.2) H2 + 300) \quad (6-6)$$

If the sender receives a RJ_TPDU or an original packet is lost, it tries to retransmit. Thus the packet is kept in the retransmission queue, an unsuccessful transmission only increases the subnet delay, as will be shown in the following section. Timeout processing has not been included in this prototype.

SECOND TRIAL:

Similarly, from Tables 5-4 and 5-6, and equations (6-1) and (6-2) we can estimate the mean execution time (in microseconds) per TSDU at level 4 for the buffer management method 3 with the following expression:

$$Ts = M[37.50 B + 402 + AK(36.0 H1 + 261) + RJ(36.0 H2 + 261)] + 218 \quad (6-7)$$

$$Tr = M[36.0 B + 448 + AK(37.5 H1 + 428) + RJ(37.5 H2 + 428)] \quad (6-8)$$

22
OF / DE

DISCUSSION:

The protocol parameters such as B, M, AK, RJ may be varied quite easily, and hence the effectiveness of different methods under different conditions readily compared. For instance, Figure 6-3 presents a comparison of the mean execution time at the transmitter for both trials, using 64 and 512 byte-TSDUs under different fragmentation schemes. Notice the execution time degradation as message fragmentation increases. Figure 6-4 compares the mean execution time to transmit correctly 1,000 512 byte-TSDUs using different acknowledgment policies.

6.2.2 THROUGHPUT CAPACITY: -LONG HAUL VS. LOCAL NETWORK-

In this section a comparison of throughput capacity between long-haul and local network is done. For the long-haul environment typical 64 and 512 byte-message transfer times for the ARPANET, as reported in [SEITZ.83], are used. For end_to_end transfer time on local area networks values presented in [MAGLAR81] for an office building scenario are used. Let us define throughput capability or maximum limit as λ . By Little's theorem, we know that the average occupancy of a buffered system (N) must equal the average delay time for the system (Delay) multiplied by the average arrival time (throughput = λ):

$$N = \lambda \cdot \text{Delay} \quad (6-9)$$

The mean number of messages in the system must be less than the window size (W), then $N < W$. Using equation (6-9) we find a bound of the maximum throughput

$$\lambda < W/\text{Delay} \quad (6-10)$$

Since:

$$\lambda \leq 1/T_s \quad (6-11)$$

$$\lambda \leq 1/T_r \quad (6-12)$$

then, from equations (6-10)-(6-12) we can define the maximum throughput in one direction to be:

$$\lambda = \text{MIN} (1/T_s, 1/T_r, W/\text{Delay}) \quad (6-13)$$

Delay, as mentioned above is the average delay time for the system or the communication subnet delay in one direction. At the transport level it is defined as:

$$\text{Delay} = M [T_n + AK (T_c) + RJ (T_c) + R (T_n)] \quad (6-14)$$

Where:

T_n = Network Service Data_TPDU transit delay.

T_c = Network Service Control_TPDU transit Delay.

R = Average number of retransmissions per TPDU.

THROUGHPUT CAPACITY FOR 64 and 512 BYTE MESSAGE:

Using equations (6-3)-(6-8) to calculate the mean execution time (in microseconds) at transmitter and receiver for 64 and 512 byte-messages, with no fragmentation, we have:

First Trial

Second Trial

TSDU = 64 Bytes (512 bits)

Ts = 3,493+ 711.8AK+ 614.8RJ Ts = 3,020+ 585 AK + 585 RJ
Tr = 3,249+ 718 AK+ 718 RJ Tr = 2,752+ 765.5AK+ 765.5RJ

TSDU = 512 Bytes (4096 bits)

Ts = 25,221+ 711.8AK+ 614.8RJ Ts = 19,820+ 585 AK + 585 RJ
Tr = 23,857+ 718 AK + 718 RJ Tr = 18,432+ 765.5AK+ 765.5RJ

To calculate the average communication subnet delay the following values are used:

Message Size (Bytes)	One-way Network Transit Delay (msec.)			
	Long-Haul (1)		Local Network (2)	
	Tn	Tc	Tn	Tc
64	262	14.77	150	10
512	709	14.77	300	10

(1) as reported in [SEITZ 83] for ARPANET.

(2) as reported in [MAGLAR81] for CSMA/CD LANs.

It remains to compute the subnet delay due to packet re-transmissions. Let P_1 be the probability that a data packet is lost or damaged, and P_2 the probability that a AK_TPDU is lost or damaged. Then the probability of success is $(1-P_1)(1-P_2)$, and the probability of failure is $L = 1-(1-P_1)(1-P_2)$. The probability that exactly K attempts are needed (i.e. $k-1$ retransmissions) is $(1-L) L^{k-1}$. So the number of retransmission per packet is $1/(1-L)$ and the expected number of retransmissions (R) one less, or $L/(1-L)$. Using the previous values and equation (6-14), we can calculate the mean subnet time:

LONG-HAUL NETWORK

$$\text{Delay}(64 \text{ Byte-TPDU}) = 262 + 14.77AK + 14.77RJ + 262 L / (1-L) \text{ msec.} \quad (6-15)$$

$$\text{Delay}(512 \text{ Byte-TPDU}) = 709 + 14.77AK + 14.77RJ + 709 L / (1-L) \text{ msec.} \quad (6-16)$$

LOCAL AREA NETWORK

$$\text{Delay}(64 \text{ Byte-TPDU}) = 300 + 10 AK + 10 RJ + 300 L / (1-L) \text{ msec.} \quad (6-17)$$

$$\text{Delay}(512 \text{ Byte-TPDU}) = 300 + 10 AK + 10 RJ + 300 L / (1-L) \text{ msec.} \quad (6-18)$$

Figures 6-5 and 6-6 present the maximum throughput capacity for long-haul and local networks versus the window size (W) for 64 and 512 byte-TPDUs. Notice that the large difference in the subnet delay for long-haul and local networks has an effect on the tuning of the flow control mechanism of the transport protocol. In a long-haul network it may be useful to use a larger window of anticipation than in a local network.

6.3 TRANSIT TIME PERFORMANCE:

The Transit time begins with the transmission of a TSDU and ends with its correct reception, i.e. $\text{TRANSIT} = T_s + \text{Delay} + T_r$. Because it ignores scheduling delay at the two ends, due to possible contention with other processing in heavily-loaded hosts or front-end processors, TRANSIT is a lower bound. Figures 6-7 and 6-8 show the components of TRANSIT for 64 byte-TSDU and 512 byte-TSDU, for both trials.

6.4 COMPARISON OF CLASS 4 AND CLASS 2

An execution time comparison between protocol class 4 and class 2, using buffer management method 3, is presented here for a situation where class 2 could be used. The network is sufficiently reliable that no packets are lost or damaged.

The mean processing time per TSDU for a protocol class 4, with $RJ=0$ and from equations (6-7)-(6-8) is given by the equations (in microseconds):

$$T_s = M [37.50 B + 402 + AK (36.0 H_1 + 261)] + 218 \quad (6-19)$$

$$T_r = M [36.0 B + 448 + AK (37.5 H_1 + 428)] \quad (6-20)$$

The mean processing time for a protocol class 2, found by taking out all the error detection and recovery functions of a class 4 transport protocol (equations (6-7)-(6-8), is:

$$T_s = M [402 + 261 AK] + 218 \quad (6-21)$$

$$T_r = M [448 + 346 AK] \quad (6-22)$$

Since no function is executed on a per-byte basis the processing time per TPDU is independent of its size.

Figure 6-9 compares the mean execution time for 512 byte-TSDUs at the transport level for class 2 and class 4 for both trials.

FIGURE 6-1. Different Buffer Management Techniques: Processing Time for Handling an Outgoing TSOU.

FIGURE 6-2. Buffer Management Processing Time to Send a 1024 byte-TSOU under Different Fragmentation Schemes.

FIGURE 6-3. Mean Execution Time per TSDU at the Transmitter (T_s) For Different Levels of Fragmentation.

FIGURE 6-4. Mean Execution Time at Level 4, to Transfer 1,000 512 byte-TSDUs under Different Acknowledgment Policies.

FIGURE 6-3. Maximum Throughput Capacity -Long Haul Vs. Local Net-Transferring 64-byte TPDU's.

FIGURE 6-4. Maximum Throughput Capacity -Long Haul Vs. Local Net-Transferring 512-byte TPDU's.

FIGURE 6-7. TRANSIT Time Components for the Transfer of a 64-byte TSDU.

FIGURE 6-8. TRANSIT Time Components for the Transfer of a 512-byte TSDU.

FIGURE 6-9. Mean Execution Time for a 512-byte TSDU, at Level 4 for Class 2 and Class 4.

CHAPTER 7

CONCLUSIONS

The work described here makes a contribution to design methods, specially, for high performance data protocols. It has shown the need to describe the protocol behaviour and environment in order to choose an optimal design. Environment issues includes: the operational environment (hardware, operating system), network characteristics, nature of the traffic. Design issues includes: one-end protocol interfaces, multiplicity of connections, buffer management and some optional functions of a transport protocol.

The data transfer phase of a transport protocol was broken down into its logical components, and execution demand, in seconds of processing, has been estimated from the demand of its components. Although the exact times will not be known until the actual code is written, an estimate can be derived from a sample design, which can be used for performance estimation.

Using sample execution times gathered from a prototype transport program predictions were made of the execution performance of four different buffer management implementation alternatives, of the maximum throughput capacity, and of the transit time of the system.

With respect to the buffer management issue, it was found that a clear performance advantage was obtained by avoiding copying data between protocol layers. The other three methods passed the data through the layers and manipulated the headers in di-

fferent ways. The preferred alternative among them depends particularly on the message size and/or level of fragmentation.

The protocol processing may be a bottleneck if the network is fast enough and the window size is large enough. For a single connection with a dedicated transport processor, the results indicated (for two example cases) the window size would have to exceed 12-60 packets depending on end_to_end delay, to make protocol processing a bottleneck. More connections would reduce the window size threshold.

The main attractions of this workload estimation are: the detection of throughput bottlenecks, understanding of the system response to unbalanced load and dynamic overload, identification of the most critical system parameters, optimization of system parameters in relation to the environment, and since the model parameters may be varied quite easily, the effectiveness of different design elements under different conditions can be readily compared.

APPENDIX A: DEFINITIONS

TRANSPORT ADDRESS: The means by which session entities can communicate using the transport service, and by which they are known to the Transport layer..

EXPEDITED TRANSPORT PROTOCOL DATA UNIT (ED_TPDU): A TPDU which, on a transport connection, will not be subject to queueing, resource restrictions or any other impediment to its rapid delivery.

EXPEDITED TRANSPORT SERVICE DATA UNITS (ED_TSDU): A TSDU which will not be subject to queueing or normal data flow control in its transport connection.

NETWORK ADDRESS: The means by which a transport entity is known to the Network layer and by which it can communicate through the network service.

NETWORK CONNECTION: A point-to-point connection between two transport entities. A network connection can be permanent or established for a limited amount of time.

NETWORK RESET INDICATION: A signal from the Network layer to the Transport layer indicating that a failure has occurred within the Network layer. An error type that can be recovered at the Transport layer without the need for disconnection of the Network connection.

NETWORK SERVICE PROVIDER: An abstract machine that models the totality of the entities providing the network service, as viewed by a transport entity.

NETWORK SERVICE DATA UNIT (NSDU): A piece of data transferred over a network connection by the Network layer for the Transport layer.

TRANSPORT CONNECTION (TC): A two way simultaneous data path between a pair of transport addresses.

TRANSPORT CONNECTION END-POINT (TCEP): A transport connection provided by the Transport layer, as seen by its user.

TRANSPORT INTERFACE DATA UNIT (TIDU):. The amount of data transferred across the transport/session interface in a single interaction.

TRANSPORT PROTOCOL DATA UNIT (TPDU): A unit of data containing control information and (possibly) user data.

TRANSPORT SERVICE USER: An abstract machine that models the totality of the entities within a single system that make use of the transport service

TRANSPORT SERVICE DATA UNIT (TSDU): A piece of data for the Session Control layer transferred by the Transport layer.

APPENDIX B: TRANSPORT SERVICE PRIMITIVES

B.1 TRANSPORT ESTABLISHMENT PHASE:

PRIMITIVE	PARAMETERS	USAGE
T_CONNECT.request	Called Address Calling Address Quality of Service Data Security	Mandatory Mandatory Mandatory User Option User Option In study
T_CONNECT.indication	Called Address Calling Address Expedited Data Service Quality of Service Data Security	Mandatory Mandatory Mandatory User Option User Option User Option In study
T_CONNECT.response	Expedited Data Service Quality of Service Data Security	Mandatory User Option User Option User Option In study
T_CONNECT.confirm	Expedited Data Service Quality of Service Data	Mandatory User Option User Option User Option

B.2 TRANSPORT DATA TRANSFER PHASE:

PRIMITIVE	PARAMETERS	USAGE
T_DATA.request	Data	Mandatory User Option
T_DATA.indication	Data	Mandatory User Option
T_EXPEDITED_DATA.request	Data	Mandatory User Option
T_EXPEDITED_DATA.indication	Data	Mandatory User Option

B.3 TRANSPORT TERMINATION PHASE:

PRIMITIVE	PARAMETERS	USAGE
T_DISCONNECT.request	Data	Mandatory User Option
T_DISCONNECT.indication	Reason Data	Mandatory Mandatory User Option

APPENDIX C: EXECUTION TIME DERIVATIONS FOR PROCEDURE-ORIENTED

INTERCOMMUNICATION PRIMITIVES

The software modules presented here perform various primitive operations required for procedure-oriented intercommunication. These modules are coded using PDP-11 assembler code. Their execution time are derived from the standard instructions times of a PDP 11/60.

The notation used through the appendix is the following:

EXECUTABLE INSTRUCTION MODULE

SUBROUTINE

EX-T

EXECUTION TIME (in microseconds).

MEM-C

NUMBER OF READ MEMORY CYCLES

EXECUTION PATH

C.1 QUEUE

To queue the node whose addr is in R1 in the queue whose head-address is in R0

INSTRUCTIONS REQUIRED:

2

QUEUE:	TST	@R0	;Is the queue empty?
	BEQ	EMPTY	
	MOV	R1,@2(R0)	;No, queue was not empty
	BR	FIN	
EMPTY:	MOV	R1,@R0	;Yes, the que was empty
FIN:	MOV	R1,2(R0)	
	CLR	(R1)	;Clear R1, correctly queued.
	RTS	PC	

EXECUTION GRAPH:

EX-T = 9.65

MEY-C = 14

C.2 DQUEUE

To remove a node from queue.

Parameters are: R0 = addr of queue head,

R1 = pointer to the dequeued node.

R2 = 1 if queue was empty, 0 otherwise.

INSTRUCTIONS REQUIRED:

DQUEUE:	TST	@R0	;Is the queue empty?
	BEQ	NONE	
	MOV	@R0,R1	;Not empty, R1 -> 1st node
	MOV	@R0,@R0	;adjust pointer of queue head
	BNE	1\$	;Did the queue is empty now?
	CLR	2(R0)	;Clean the tail pointer
1\$:	CLR	R2	;Indicate queue was not empty
	RTS	PC	
NONE:	MOV	#1,R2	;Indicate queue empty
	RTS	PC	

EXECUTION GRAPH:

C.3 DSPTCH

If RTR1 queue is not empty

THEN remove 1st job in RTR1 & set it running

ELSE remove 1st job in RTR2 & set it running.

INSTRUCTIONS REQUIRED

```

DSPTCH: MOV    #RTR1,R0      ;Dequeue from RTR1
        JSR    PC,DQUEUE
        TST    R2           ;Was RTR1 empty
        BEQ    1$
        MOV    #RTR2,R0      ;Yes, dequeue from RTR2
        JSR    PC,DQUEUE
1$:      MOV    R1,RUNPTR
        MOV    PSTACK(R1),SP ;Restore SP
        CLR    PSTACK(R1)    ;Indicate process running
  
```

EXECUTION GRAPH:

C.4 WAIT

To WAIT on a semaphore whose ADDR is in R0.

INSTRUCTIONS REQUIRED

```

WAIT:  MOV    (SP),R0      ;Get addr of the semaphore
        DEC    4(R0)      ;Decrement counter
        BLT    1$         ;Counter non-negative
        JMP    OUT        ; Calling process continues
1$:     MOV    RUNPTR,R1   ;Counter negative
        MOV    SP,PSTACK(R1) ;Calling process delayed in
        MOV    #3,PSTATE(R1)
        JSR    PC,QUEUE
        JMP    DSPTCH
  
```

EXECUTION GRAPH:

C.5 SIGNAL

To SIGNAL a semaphore whose ADDR is in R0.

INSTRUCTIONS REQUIRED

SIGNAL:	MOV	(SP),R3	;Get semaphore address
	TST	4(R3)	;If counter negative
	BGE	1\$	
	MOV	R3,R0	;Then release the 1st
	JSR	PC,DQUEUE	;process delayed in
	CMP	PRIOR(R1),#1	;sem. queue and place
	BNE	2\$	;it in the appropriate
	MOV	#RTR1,R0	;RTR queue.
	MOV	#1,PSTATE(R1)	; In RTR1 queue
	JSR	PC,QUEUE	
	BR	1\$	
2\$:	MOV	#RTR2,R0	; In RTR2 queue
	MOV	#2,PSTATE(R1)	
	JSR	PC,QUEUE	
1\$:	INC	4(R3)	;Increment counter
	JMP	OUT	

EXECUTION GRAPH:

APPENDIX C

C-7,

C.6 MONITOR SYNCHRONIZATION PROCEDURES

C.6.1 ENTER

To serialize the request for service and admitting only one processes at the time.

INSTRUCTIONS REQUIRED

HIGH LEVEL INSTRUC.

```
Procedure ENTER (var c:ctrl);
Begin
    WAIT(c.gate);
End;
```

PDP11 ASSEMBLER INSTRUC.

```
.GLOBL ENTER
ENTER:
    MOV    2(6),-(6)
    JSR    $7,WAIT
    MOV    (6),2(6)
    ADD    $2,$6
    RTS    $7
```

EXECUTION GRAPH:

C.6.2 LEAVE

INSTRUCTIONS REQUIRED

HIGH LEVEL INSTR.

PDP11 ASSEMBLER INSTRUC.

```

Procedure LEAVE(var c:ctrl)
Begin
  With c do
    If HIPRIWAIT > 0 then

      Begin
        HIPRIWAIT := HIPRIWAIT-1;
        SIGNAL(hipri);

      End
    else SIGNAL(GATE);

  End;

```

```

.GLOBL LEAVE
LEAVE:
MOV     2(6),%3
TST     6(3)
BGT     L1
JMP     L2

L1:     DEC     6(3)
        MOV     %3,-(6)
        MOV     %3,%0
        ADD     %8,%0
        MOV     %0,-(6)
        JSR     %7,SIGNAL
        MOV     (6)+,%3
        JMP     L3

L2:     MOV     %3,-(6)
        MOV     %3,-(6)
        JSR     %7,SIGNAL
        MOV     (6)+,%3
        MOV     (6),2(6)
        ADD     %2,%6
        RTS     %7

L3:

```

EXECUTION GRAPH:

C.6.3 SLEEP

HIGH LEVEL INSTRU.

PDP11 ASSEMBLER INSTRU.

Procedure SLEEP (var e:even)

Begin

With e do begin
 evwait := evwait + 1;
 LEAVE(ctrllink^);

WAIT(evensem);

WAIT(ctrllink^.hipri);

End;

SLEEP: .GLOBAL SLEEP

```

MOV    2(6),%3
INC    2(3)
MOV    %3,-(6)
MOV    %3,-(6)
JSR    %7,LEAVE
MOV    (6)+,%3
MOV    %3,-(6)
MOV    %3,%0
ADD    #4,%0
MOV    %0,-(6)
MOV    %3,-(6)
MOV    @%3,%0
ADD    #8,%0
MOV    %0,-(6)
JSR    %7,WAIT
MOV    (6)+,%3
MOV    (6),2(6)
ADD    #2,%6
RTS    %7

```

EXECUTION GRAPH:

C.6.4 AWAKEN

INSTRUCTIONS REQUIRED

HIGH LEVEL INSTRUC.

```

Procedure AWAKEN (var e:event)
Begin
  With e do
    If evwait > 0 then

      Begin
        evwait := evwait-1;
        ctrllink^.hipriwait:=
          ctrllink^.hipriwait+1;

        SIGNAL(evsem);

      End

    End
  End;
End;

```

PDP 11 ASSEMBLER INSTRUC.

```

      .GLOBAL AWAKEN
AWAKEN:
      MOV     2(6),%3
      TST     2(3)
      BGT     L4
      JMP     L5

L4:
      DEC     2(3)
      MOV     %3,%0
      MOV     %3.1
      MOV     6(1),%1
      INC     %1
      MOV     %1,6(0)
      MOV     %3,-(6)
      MOV     %3,%0
      ADD     %4,%0
      MOV     %0,-(6)
      JSR     %7,SIGNAL
      MOV     (6)+,%3

L5:
      MOV     (6),2(6)
      ADD     %2,%6
      RTS     %7

```

EXECUTION GRAPH:

REFERENCES

- [ANDLER81] Andler, S., D. Daniels and A. Spector, "On Enhancing Local Network Communication Devices", Local Networks for Computer Communications, North-Holland Publishing Co. 1981, pp 191-204.
- [BOCHMA83a] Bochman, G., "Formal Description Techniques for Protocols", Final Report for DOC contract No. OSU82-00218, March 1983.
- [BOCHMA83b] Bochman, G., "Protocol Testing and Implementation", Final Report for DOC contract No. OST82-0092. March 1983.
- [BUHR 81] Buhr, R.J. and D.A. MacKinnon, "The Transport Layer in Open System Interconnection: A study of Issues", Department of Communications, Ottawa, Canada, 1981.
- [BUX 80] Bux, W., "Local Area Networks: A performance Comparison", Local Area Network Workshop, August 1980, Zurich, Switzerland.
- [CLARK 83] Clark, D., "Practical Considerations in Network Protocols: Implementation and Design", Data Comm. Symposium 1983, Cape Codd, Mass., Oct. 1983.
- [CERF 78] Cerf, V.G., "DARPA Activities in Packet Network Interconnection", Proceedings of the NATO Advanced Studies Institute Conference on Network Intercon. Bonas, France, September 1978.

- [COHEN 79] Cohen, D. and J. Postel, "On Protocol Multiplexing" Proceedings, Sixth Data Communications Symposium, Nov. 1979, pp 75-81.
- [ECMA 81] European Computer Manufacturers Association, "Standard ECMA-72 Transport Protocol".
- [GERLA 80] Gerla, M. and L. Kleinrock, "Flow Control, A Comparative Survey", IEEE Trans. on Comm., COM-28, pp. 553-574, 1980.
- [GREEN 79] Green, P.E., "An Introduction to Network Architecture and Protocols", IBM System Journal, Vol 18, No.2, 1979, pp 202-222.
- [GRIFFI84] Griffiths, C., "Protocol Software Portability", M.C.S. Thesis, Carleton University, 1984.
- [HOARE 74] Hoare, C.A., "Monitors: An operating System Structuring Concept", Communications of ACM, Vol. 17, No. 10, October 1974, pp. 549-557.
- [ICHBIA76] Ichbiah, J.D. et al, "Rational for the Design of the ADA Programming Language, ACM SIGPLAN Notice Vol. 14, No. 6, June 79, Part B.
- [ISO 79] International Organization for Standardization, "Reference Model Open System Interconnection", Document ISO/TC 97/SC 16 N 227, June 1979.

REFERENCES

R-3

- [ISO 82] International Organization for Standardization, "Transport Protocol Specification", Document ISO/TC 97/SC 16 N 1169, June 1982.
- [ISO 83] International Organization for Standardization, "Network Service Definition", Document ISO/TC 97/SC 16 N 2744, May 1983.
- [KERNIG78] Kernigham B.W. and D.M. Ritchie, "The C Programming Language", Prentice-Hall, Inc., 1978.
- [MAGLAR81] Maglaris, B., T. Lissack and M. Austin, "End-to-End Delay Analysis on Local Area Networks: An office Building Scenario", Proc. NTC'81, New Orleans, Nov. 1981, Session A.2.3.
- [MANNIN78] Manning, E.G., "On Datagram Service in Public Packet Switched Networks", Computer Networks, Vol. 2 1978, pp 79-83.
- [McGILT83] McGilton, H. and R. Morgan, "Introducing the UNIX System", McGraw-Hill Book Co., 1983.
- [McQUILL78] McQuillan, J.M., "Message Addressing Modes for Computer Networks", Computer Communications Networks, COMPCON 78 Proceedings, Sept. 1978.
- [NBS 83] National Bureau of Standards, U.S. Department of Commerce "Proposed Federal Information Processing Standard", Vols. 1-6, 1983.

REFERENCES

R-4

- [OHBA 78] Ohba, H., S. Yoshitake, S. Mutoh and T. Nishimura, "End-to-End Protocol Based on CCITT X.25 and its Implementation", Proc. of the 4th. Inter. Conf. on Comp. Comm., Kyoto, Sept. 78.
- [PIATKO79] Piatkowski, F., "A Formal Definition of the ISO-ANSI, System Approach", Proceedings, 1st. Inter. Conf. on Distri. Comp. sys., Oct. 1979, pp 79-94.
- [PIATKO82] Piatkowski, F., L. Ip and D. Hee, "State Architecture Notation and Simulation: A Formal Technique for the Specification and Testing of Protocol Systems", Computer Networks journal 6, 1982, pp 397-418.
- [POH 79] Poh, S., P.O. Stoneburner and D.C. Wood, "A Performance Study of a Network Front End", Proc. Sixth Data Communication Symposium, Nov. 1979, pp 126-136.
- [POUZIN79] Pouzin, L. and H. Zimmerman, "A tutorial on Protocols", Proceedings of the IEEE, Nov. 1978, pp 1346-1370.
- [RAGHUN83] Raghunathan, K.S., J.A. Barchanski and G. Bochmann, "Relationship Between Performance Parameters for Transport and Network", ACM SIGCOMM'83 Symposium, March 1983, pp 129-135.
- [REISER79] Reiser, M., "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control", IEEE Trans. on Comm., COM-27, pp.1199-1209, 1979.

- [SEITZ 83] Seitz, N.B., D.R. Wortendyke and K.P. Spies, "User-Oriented Performance, Measurements on the ARPANET", IEEE Communication Magazine, Sept. 1983.
- [SEVCIK78] Sevcik, P.J. and P.J. Nichols, "Packet Transport Protocol Design Consideration", Proceedings, Computer Networking Symp., 1978, pp 129-137.
- [SHOCH 80] Shoch, J.F. and J. Hupp, "Performance of an Ethernet Local Network", Proceedings of 20th Computer Soc. Intl. Conf., San Francisco, California, 1980.
- [TOBAGI79] Tobagi, F.A. and B. Hunt, "Performance Analysis of Carrier Sense Multiple Access with Collision Detection", Proceedings of the LACN Symposium, May 1979, pp.217-245.
- [WATSON80] Watson, W.B., "Performance in Contention Bus Local Network Interconnection", Lawrence Livermore National Laboratory Report, Livermore, Cal. 1980.
- [WONG 79] Wong, G., "Modelling Methodology for Computer System Performance", Oriented Design Methodology, North-Holland Pub. Co., 1979.
- [WOODSI83] Woodside, C.M., J.R. Montealegre and R.J. Buhr, "A Performance Model for Hardware/Software Issues in Computer-Aided Design of Protocol Systems", Submitted for publication, Oct. 1983.
- [YUM 82] Yum, T.P. and C. Dou, "Design Algorithms for Buffer Allocation Strategies in a Computer Network Node", IEEE GLOBECOM'82, Miami, Fla., Nov. 1982, Session B4.4.

END

12-06-86

FIN