T

**PM-1 3½"x4" F. PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT**

PRECISIONSM RESOLUTION TARGETS

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform 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.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme 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.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

***Evaluation of Bypass Architecture for High-Speed Bulk
Data Transfer through deeply layered protocol stacks***

by

Yong Hua Thia, BSc, MSc, DIC

A Thesis submitted to the Faculty of Graduate Studies and Research in partial
fulfillment of the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Electrical Engineering,
Faculty of Engineering,
Department of Systems and Computer Engineering,
Carleton University,
Ottawa, Ontario, Canada, K1S 5B6
March 5, 1994

© Yong Hua, Thia 1993

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-92944-8

Canada

Name YONG HUA, THIA

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

0544

U·M·I

SUBJECT TERM

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
Ancient 0289
Linguistics 0290
Modern 0291
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical 0310
Business Administration 0272
General 0770
Accounting 0454
Banking 0338
Management 0385
Marketing 0501
Canadian Studies 0503
Economics 0505
General 0508
Agricultural 0509
Commerce-Business 0510
Finance 0511
History 0358
Labor 0366
Theory 0351
Folklore 0578
Geography
Gerontology
History
General

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0616
International Law and Relations 0617
Public Administration 0814
Recreation 0452
Social Work 0626
Sociology 0627
General 0938
Criminology and Penology 0631
Demography
Ethnic and Racial Studies
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0374
Cell 0329
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences 0566
General 0300
Audiology 0992
Chemotherapy 0567
Dentistry 0350
Education 0769
Hospital Management 0758
Human Development 0982
Immunology 0564
Medicine and Surgery 0347
Mental Health 0569
Nursing 0570
Nutrition 0380
Obstetrics and Gynecology
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics
Physics 0605
General 0986
Acoustics
Astronomy and Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and High Energy 0798
Fluid and Plasma 0759
Molecular 0409
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463

Applied Sciences

Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General 0538
Aerospace 0539
Agricultural 0540
Automotive 0541
Biomedical 0542
Chemical 0543
Civil 0544
Electronics and Electrical 0348
Heat and Thermodynamics 0545
Hydraulic 0546
Industrial 0547
Marine 0794
Materials Science 0548
Mechanical 0743
Metallurgy 0551
Mining 0552
Nuclear 0549
Packaging 0765
Petroleum 0554
Sanitary and Municipal 0790
System Science 0428
Geotechnology
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

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

**"Evaluation of Bypass Architecture for High-Speed Bulk
Data Transfer through deeply layered protocol stacks"**

submitted by

Yong Hua Thia, BSc, MSc, DIC

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Chair, Department of Systems and Computer Engineering
Professor Samy A. Mahmoud

Thesis Supervisor
Professor Murray C. Woodside

External Examiner
Dr. Martina Zitterbart

Carleton University
March 2nd 1994

ABSTRACT

Techniques for handling high data throughput in end-systems attached to data networks are the subject of this thesis. The advent of Fibre Optic technology which offers high bandwidth and low bit error rates, has shifted the *performance bottleneck* in data communications from the channel to the processing in the end-points of the system. To alleviate the end-system bottleneck one may consider new protocols, improved software implementation of existing protocols, parallel processing techniques, special protocol structures and hardware assist by offloading all or part of the protocol functions to an attached processor or adapter.

In this thesis, speedup in protocol processing is accomplished by using a combination of approaches based on general software performance engineering principles. A novel concept for protocol implementation, which is a generalization of Jacobson's "Header Prediction" algorithm for TCP/IP is introduced. Bypassing identifies a fast path for bulk data transfer, by which frequent simple operations are found and optimized. We propose and evaluate three different methods for bypassing. In the first approach, throughput performance is increased by pure software improvements due to bypass. The second approach uses dedicated VLSI hardware for offloading protocol functions in the bypass path. A feasibility study was conducted using the industry standard VHDL language to model the bypass chip which we call ROPE (Reduced Operation Protocol Engine). Results show that it is feasible to implement the bypass path for the Session and Transport layers of OSI protocols in VLSI with current gate array technology, and that ROPE is capable of supporting a very high data rate — 313 Mbps with TP4 checksum. Finally, in the third approach, a scalable packet-level architecture is proposed and evaluated. It uses digital signal processing components which are readily available. A simulation model with realistic instruction execution times was performed, and its results were validated with an analytical model. With POBA, a system executing the three layers (transport, session and presentation) can scale up to speeds close to the host memory bandwidth.

ACKNOWLEDGMENTS

This research was supported in large part by the Telecommunications Research Institute of Ontario (TRIO).

The debt of gratitude that a Phd student owes his advisor at the end of a project this size is enormous. I am greatly indebted to Prof. Murray Woodside for his excellent guidance, continued encouragement, and generous support, as well as his extraordinary patience in improving my writing and communication skills.

Discussions with Craig Scratchley, Erik Dravnieks, Govindachari Raghunath, Reza Etemadi, Srinivas Vunnava, Weimin Ma, Yao Li and many others have been both invigorating and informative. Greg Franks helped me get started on the testbed implementation and Curtis Hrischuk was always there to help with questions pertaining to PARASOL.

I would also like to thank Bell-Northern Research which provided the VHDL tools and facility used in this study. I am particularly indebted to Dr. Simon Curry, Hemi Thakar, Dr. Parviz Yousefpour, Bernard Doray, Mike Majid and Mustapha El Ahla who helped with the use of their tools and made my stay at BNR an enjoyable and memorable one.

I am grateful to all my friends and fellow students in the Systems and Computer Engineering department. In particular, I enjoyed playing squash and going on ski trips with Norm Lo, Francois Vien, Robert Dussault and Quyen Vi Bo. Special thanks are due to the office and support staff, Naren Mehta, John Pope, Dave Sword, Danny Lemay, Darlene Hebert, Elena Keen, Vivienne Gilchrist, and others who assisted in one way or another.

Last but not least, I am deeply grateful to my parents for their support and encouragement. My wife, Irene and son Brandon, endured years of sacrifice and hardships to see me through this program. I owe my family so much that I could find no words to express my deepest gratitude.

Contents

ABSTRACT	iii
ACKNOWLEDGMENTS	iv
List of Figures	viii
List of Tables	x
1 Introduction	
1.1 The problem statement and motivation	1
1.2 The proposed solution	3
1.3 Summary of contributions	6
1.4 Thesis Outline	7
2 Background	
2.1 Introduction	8
2.2 OSI Architecture and layering	9
2.2.1 Why Layering?	10
2.2.2 Layering Principles	11
2.3 Overview of the Seven Layers of the OSI Architecture	13
2.4 Why are protocol implementations slow?	15
2.4.1 Processing and data path of a typical application PDU	15
2.4.2 Host Architecture	18
2.4.3 Adapter	18
2.4.4 Host/Adapter interface	19
2.4.5 Operating system support	20
2.4.6 Buffer Management	21
2.4.7 Timer Management	22
2.4.8 Software Implementation	23
2.4.9 Protocol structure and mechanism	23
2.5 Software Performance Engineering Principles	24
3 Review of the State of the Art in Performance Improvement of Upper Layer Protocols	
3.1 Introduction	27
3.2 Lightweight Transport Protocols and their mechanisms to support high-speed networks	28
3.2.1 Summary of Transport Service Functions that affect performance of high-speed networks	30
3.2.1.1 Connection Management	31
3.2.1.2 Flow Control and Acknowledgment	32
3.2.1.3 Error Handling	33
3.2.2 Lightweight Transport Protocols	33
3.3 Special Implementation Techniques	36
3.3.1 Header Prediction	38

3.4	Hardware Implementation	39
3.5	Parallel Hardware Implementation	40
3.6	Special Architecture or Special Protocol Structures	43
3.6.1	Integrated Layer Processing	44
3.6.2	AXON	44
3.6.3	HOPS	46
3.6.4	Flexible High-Performance Communication Subsystem	46
3.7	High-Speed Presentation Layer	47
4	Problem Statement and Proposed Solution	
4.1	Characterization of Gbps processing requirement	48
4.2	Goals of this thesis	52
4.3	State-of-the-art techniques from the SPE point of view	54
4.4	Characteristics of the Proposed Solution	56
4.5	Proposed Optimistic Principle	58
5	The Bypass Concept and Software bypass	
5.1	Bypass Concept	60
5.2	Bypass Architecture and its Implementation	63
5.2.1	Send Bypass Test	63
5.2.2	Receive Bypass Test	64
5.2.3	Bypass Stack	65
5.2.4	Shared Data	65
5.2.5	Isolating Data Transfer phase from Connection Establishment and Release phase	65
5.3	Multiple-layer bypass	66
5.3.1	Difficulties of multiple-layer bypass	67
5.4	Bypass without window flow control	68
5.5	Bypass Algorithm with window flow control	70
5.6	Experimental Setup	71
5.6.1	Software structure	71
5.7	Results of Bypass Implementation	74
5.8	Summary	77
6	A Reduced Operation Protocol Engine (ROPE): Design and Implementation of Bypass chip	
6.1	Introduction to VHDL concepts	79
6.1.1	Design	80
6.1.2	Architectural Description	81
6.1.3	Experimental Setup	81
6.1.4	Behavioral description	84
6.1.5	Extensions to include major procedures for Transport Class 4 (Implemented)	87
6.1.5.1	OSI Checksum	87
6.1.5.2	Timers	88
6.1.5.3	Retransmission and Resequencing	88

6.2	Results	89
6.3	Extensions to support Session BAS and BSS, and the Presentation layer (Not Implemented)	90
6.3.1	Session BSS and BAS	91
6.3.2	Presentation Layer processing	93
6.4	Summary	94
7	POBA -- Parallel Optimistic Bypass Architecture	
7.1	Description of POBA	95
7.1.1	Hardware Architecture	96
7.1.2	Sequence of Operations	97
7.1.3	Protocol processing at each PPN Node	100
7.1.4	Advantages of POBA	100
7.2	Characterization of processing requirements and throughput analysis of POBA	101
7.2.1	Workload Analysis for POBA	102
7.2.2	Performance bounds	105
7.2.3	Scalability	108
7.3	An implementation using standard off-the-shelf DSP microprocessors	110
7.3.1	Proposed Hardware for the adapter	110
7.3.2	Task Architecture	110
7.3.3	Simulation technique	114
7.4	Simulation results	117
7.5	Simulation versus analytical results	122
7.6	Bypass Issues	126
7.7	Predicted performance	127
7.8	Summary	128
8	Conclusions and Future Work	
8.1	Conclusions	130
8.2	Advantages and What it solves	132
8.3	Future Work	134
8.3.1	New Applications for Bypass	135
8.3.2	Applying performance principles for fast connection setup in connection-oriented protocols	138
	List of Abbreviations	142
	Bibliography	145
Appendix A	Pseudo Code for bypass	159
Appendix B	TP4 Checksum	163
Appendix C	ASN.1 Basic Encoding Rule for Integers	165
Appendix D	Schematic Diagram Generated by SYNOPSIS synthesis tool (See 6.1.3)	168
Appendix E	Timeline Snapshot (See 7.3.3)	169
Appendix F	Simulation Results for POBA (See 7.4)	170

List of Figures

Figure 1	Open System Interconnection Reference Model	10
Figure 2	Layering Interface	12
Figure 3	Layer Interface	13
Figure 4	Data Movement in End-Systems	17
Figure 5	State of the Art Review	29
Figure 6	Zitterbart: Structural and Functional Parallelism	41
Figure 7	Jain Schwartz and Bashkow: Packet level Parallel processing of TP4	42
Figure 8	Sterbenz and Parulkar: AXON Block Diagram	45
Figure 9	Haas: HOPS Architecture	46
Figure 10	Effect on throughput by varying X and B	50
Figure 11	Processor requirement for 1 Gbps protocol processing	51
Figure 12	Time Sequence Diagram	61
Figure 13	Finite State Machine of TP0/TP2	62
Figure 14	Bypass Architecture	64
Figure 15	Critical path taken by bypassed packets	68
Figure 16	Sender and Receiver Header Templates	69
Figure 17	ACK TPDU	70
Figure 18	Header Templates	71
Figure 19	Experimental testbed	72
Figure 20	Procedural Flow Chart of Protocol Testbed	74
Figure 21	Software and Hardware processing path of a non-bypass versus a bypass system	80
Figure 22	Block Diagram of VLSI bypass system	82
Figure 23	Design flow diagram	84
Figure 24	Organization of internal bypass chip memory	86
Figure 25	Activity management, Dialogue units and Synchronization points	92
Figure 26	Send Bypass Stack Parallel Architecture	98
Figure 27	Bus Utilization	103
Figure 28	Performance Bounds	107
Figure 29	Effects of varying X and B_{cp} on throughput Bounds	108
Figure 30	Scalability of POBA with packet size (P) and PPN workload (XP)	109
Figure 31	Software task architecture	111
Figure 32	3D plot of Throughput versus X and N (1 Kbyte)	117
Figure 33	Throughput versus N (1 Kbyte)	118
Figure 34	Throughput versus X (1 Kbyte)	118

Figure 35	Average PPN utilization versus N (1 Kbyte)	119
Figure 36	Average PPN utilization versus N (1 Kbyte)	120
Figure 37	Control Processor Utilization versus N (1 Kbyte)	121
Figure 38	Control Processor Utilization versus N (1 Kbyte)	121
Figure 39	POBA with 2 Processors	122
Figure 40	POBA with 4 Processors	123
Figure 41	POBA with 8 Processors	123
Figure 42	POBA with 10 Processors	124
Figure 43	Throughput versus N — Simulation results compared with analysis	125
Figure 44	Throughput versus N — Simulation results compared with analysis	125
Figure 45	Throughput versus message length — Simulation results compared with analysis	126
Figure 46	Effect of switch in processing paths, causing the bypass path to be flushed.	127
Figure 47	Extrapolation of throughput bounds for improved technology	128
Figure 48	Optimistic Connection Setup	140
Figure 49	POBA under various Workloads	170
Figure 50	POBA with multiple active Connections	171
Figure 51	Effect of switch in processing path for POBA with $N = 8$.	171
Figure 52	3D plot of Throughput versus X and N (4 Kbytes) . . .	172
Figure 53	Throughput versus X (4 Kbytes)	172
Figure 54	Throughput versus N (4 Kbytes)	173
Figure 55	3D plot of Control Processor Utilization versus X and N (4 Kbytes)	174
Figure 56	Control Processor Utilization versus X (4 Kbytes)	174
Figure 57	Control Processor Utilization versus N (4 Kbytes)	175
Figure 58	3D plot of PPN Utilization versus X and N (4 Kbytes) .	176
Figure 59	PPN Utilization versus X (4 Kbytes)	176
Figure 60	PPN Utilization versus N (4 Kbytes)	177

List of Tables

Table 1	Comparison of Lightweight Protocols	37
Table 2	Common procedures for Presentation, Session and Transport layers during data transfer phase	52
Table 3	Bypass of Session/TP0	76
Table 4	Bypass of Session/TP2	77
Table 5	Throughput Performance and gate count of bypass VLSI chip (ROPE)	90
Table 6	Partitioning of protocol functions to host, PPNs and the Control processor	102

Chapter 1

Introduction

Gigabit networks are currently the subject of much research attention [123]. Applications for future Gigabit per second (Gbps) networks will probably include multimedia communications, distributed high-speed real-time graphics and support for the bulk transfer of scientific data, such as that of the High Energy Physics community [76].

1.1 The problem statement and motivation

The advent of Fibre Optic technology which offers high bandwidth and low bit error rates, has shifted the *performance bottleneck* in data communications from the channel to the processing in the end-points of the system. The heavy processing load is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. The operating system overhead includes *non-protocol related* processing like context switching, interrupt handling, movement of data across layer boundaries and process scheduling associated with handling several concurrent stream of events. The protocol complexity is partly historical [105, 183] and partly to provide very flexible standard capabilities. The per-octet processing is found in checksums and presentation processing. Conventional transport protocols like TCP [27, 31, 54, 198] and OSI TP4 [42, 53], which include complex functionality for connection establishment, release and exception handling are considered too inefficient for high speed communications and too complicated to be implemented in hardware. The problem is even worse for the Session [35, 97, 98] and Presentation [99, 100] layers which have a larger set of control Protocol Data Units (PDUs), each with varying

data formats and optional control fields. For bulk data transfer, per-octet operations like presentation encoding/decoding and checksum processing will significantly affect throughput performance.

The performance of upper layer protocols also depend on the hardware it is executing on. This includes the host processor, the memory, the bus and the network adapter. Current high performance workstation architectures have been optimized for data processing rather than data movement [51]. The RISC based processors common in these workstations typically rely on cache memory to prevent "bubbles" in the processor pipeline. As host processing speed continues to outpace improvements in memory bandwidth, the penalty for access from main memory due to cache misses increases [71, 87]. Also, as the network bandwidth approaches the memory bandwidth, it is important to keep data movement on the workstations down to the minimum, especially for bulk data transfer. The ideal situation will be to transfer the application data units directly from the user space to the adapter buffer, with minimal host intervention. This suggests that some protocol functions must be offloaded.

The key problems associated with offboard processing of protocols include:

- It is often difficult to partition the functionality of protocols in a clear-cut way for off-board processing. Poor partitioning of functions lead to a complex protocol for the host software to communicate with the adapter, which may offset the potential gain in performance due to offloading [174].
- It is not sufficient to offload the protocol functions and hope to achieve a dramatic improvement in performance. Non-protocol related processing is a large part of the total load, as shown in [201]. Examples include interrupt handling, context switching and data copying at layer boundaries in deeply layered protocol stacks.
- It is important to consider multiple layers together [43]. However, it increases the complexity of the adapter especially if the full protocol logic of all the layers are offloaded [28, 106].

- The hardware in the adapter might range from several microprocessors to single-chip VLSI implementations. Probably because of the complexity of existing protocols, VLSI [118] implementation above the data link layer has been disappointing so far. The choice of hardware depends largely upon the complexity of the functions supported by the adapter. In [13] where the entire transport protocol layer is offloaded or in [106] where the full protocol stack can be offloaded, general purpose microprocessors are used. In [59], dedicated VLSI chips are used to support TCP checksums. Also, some newer lightweight transport protocols are designed for VLSI implementation [11, 41].

1.2 The proposed solution

This thesis uses a combination of approaches based on general software performance engineering principles to solve the problem addressed in the previous section. It employs the *protocol bypass* concept [210] which is a generalization of Jacobson's "Header Prediction" algorithm [107] for TCP/IP. Bypassing identifies a fast path for bulk data transfer, by which frequent simple operations are found and optimized. It addresses the problems associated with off-board processing by selecting a particular set of functions which are complete in themselves for offloading. The primary goal of this thesis is to verify and extend the bypass concept in order to deliver very high throughput rates (up to the host bus/memory bandwidth) to the application program while still using standard "heavyweight" protocols. By this, we hope that we can achieve first, a usable engineering approach, and second, a deeper understanding of performance issues which would guide future designs of protocol and host system architecture.

This thesis also addresses the following questions associated with bypassing:

- How to improve performance in the bypass path?
- How to minimize bypass test overhead and synchronization between the bypass path and the standard protocol stack?

- What are the problems associated with multiple-layer bypass?
- Partitioning of functionality for bypass and its impact on performance.

Three approaches are proposed each of which serves to address one or more of the above questions. In the first approach, throughput performance is increased by implementing changes to software alone. This study used an experimental protocol testbed, an algorithm for multiple-layer bypass with window flow control, and an efficient logic for the bypass test [196]. It verified the bypass logic and concept.

The second approach proposes the use of dedicated VLSI hardware for offloading protocol functions in the bypass path. Since the bypass system identifies a simple subset of protocol processing, it is more amenable to VLSI implementation than an entire protocol. This results in an architecture that offloads critical functions that are inefficiently handled by current host systems to dedicated VLSI hardware, leaving only a little header processing and a DMA transfer to the adapter for the host, and requiring only a simple command protocol. A feasibility study was conducted using the industry standard VHDL language to model the bypass chip, which is called ROPE (Reduced Operation Protocol Engine) [195]. A behavioral model of the complete bypass system and a structural model of ROPE were constructed. Results show that it is both feasible to implement the bypass path for the Session and Transport layers of OSI protocols in VLSI with current gate array technology, and that it is capable of supporting a very high data rate.

Finally, in the third approach, a *scalable* packet-level Parallel Optimistic Bypass Architecture (POBA) [194] was designed and evaluated. Current off-the-shelf DSP microprocessors are used to evaluate the architecture experimentally with very realistic simulation. Its results are compared with an analytical model. With POBA, a system using a small number of available DSP components can execute the three layers (transport, session, presentation) at speeds close to the host memory bandwidth [194]. The POBA architecture is less complex, with a significant reduction in processor contention for shared connection information. Hence its efficiency for parallel implementation is significantly

higher compared to architectures that implement the full protocol logic [21, 106].

The bypass concept does not address the processing requirement for fast connection setup. We believe that the speed in processing connection packets is not the main issue in high-speed networks, but the number of message exchanges required between two application entities before a connection can be established.

Recent projects on high-speed protocol processing have explored various approaches, each of which has a connection to this research:

- *New "lightweight" protocols* — Our approach can make standard protocols nearly as fast as these lightweight protocols, at least for bulk data transfer. In bypass, lightweight data transfer can be associated with a complete or heavyweight protocol for connections.
- *Special software implementation techniques* — Bypass reduces inter-layer overhead when it is extended to a multiple-layer protocol stack. Inter-layer processing overhead includes queue and buffer management, context switching and movement of data across layers, all of which are a significant overhead in communications processing. The advantage of this work is the ability to integrate the critical processing functions of a deeply layered protocol stack without a major redesign of current standards.
- *Hardware implementation* — The proposed architecture moves all the per-octet processing to the adapter, leaving only a little header processing and a DMA transfer to the adapter for the host, and requiring only a simple command protocol.
- *Parallel hardware implementation* — The bypass approach is a good way to organize the processing within the parallel hardware architecture described by Jain *et al* [111] and Goldberg *et al* [82], and will significantly enhance their proposal. Bypass reduces the complexity of protocol processing resulting in a significant reduction in processor contention for shared connection information.
- *Special Architecture and protocol structures* — Multiple-layer bypass merges adjoining layers to form a "super-efficient-stack" during the data transfer phase. The

functionality of these layers can then be organized for optimal performance.

The main difference of this work, compared with other ideas in the literature is that it addresses the processing requirements of a **multiple-layer** protocol stack by efficiently offloading per-octet **protocol related** operations and minimizing the key **non-protocol related** per-packet operations. It succeeds by isolating functions that are typically handled inefficiently by the host and its operating environment. It allows the system to transfer the application data units directly from the user space to the adapter buffer, with minimal host intervention. Although standard protocols are used to demonstrate the feasibility of this approach, this work is not restricted to a particular protocol mechanism. For example, lightweight protocols can also be incorporated for bypassing. It will result in a significant improvement in performance as multiple-layer bypass reduces interlayer processing overhead which is more significant than pure protocol processing. Apart from processing requirements, newer lightweight protocol mechanisms like rate control for high-speed congestion control and lightweight connection mechanisms [64] which minimize the amount of messages exchanged between two application entities, will also be required.

1.3 Summary of contributions

The authors original contributions are:

- Validation and extension of bypass to cover flow control, multiplexing, the session layer, and the presentation layer. Experimental study of software bypass is based on an actual functional stack which includes all the phases of a connection (see chapter 5).
- Another design for a chip implementation, the Reduced Operation Protocol Engine (ROPE) is given. The result shows that the bypass path of the transport, session and some presentation conversion procedures can be implemented on a chip (see chapter 6).

- A design is given for a parallel processing adapter (POBA) using commercial processors. It is scalable up to the host bus/memory bandwidth (see chapter 7).
- As a secondary contribution, a new software performance engineering principle is identified, called the Optimistic principle. The optimistic approach shows promise for increasing network performance as demonstrated by the header prediction algorithm [107].

1.4 Thesis Outline

Chapter 2 includes background material on the OSI layered model, its protocols and outlines the problems associated with end-system protocol processing performance. It also gives a brief introduction of the general Software Performance Engineering Principles. In Chapter 3, a comprehensive review of the state-of-the-art research activity in high-speed upper layers protocol processing is presented, highlighting the major contributions in this field to date. The problem statement of this thesis together with a summary of the proposed technique in comparison to existing work is given in chapter 4.

Chapter 5 describes the bypass concept, its architecture and implementation. An algorithm for bypass with window flow control and some results conducted on an experimental software testbed is presented here. Chapter 6 describes the design and implementation of a VLSI bypass chip called ROPE (Reduced Operation Protocol Engine). Chapter 7 proposes the Parallel Optimistic Bypass Architecture (POBA) which extends the bypass concept further, followed by some conclusions and suggestions for future research in chapter 8.

Chapter 2

Background

2.1 Introduction

In the early to mid-seventies, many computer manufacturers designed networking products which allow different models from the same manufacturer to communicate. IBM came out with Systems Network Architecture (SNA) while DEC used its proprietary Digital Network Architecture (DNA) and many others [188]. They were all similar in employing a concept called the layered architecture. The absence of a universal *standardization* to address this problem at that time has led the manufacturers to continue to develop their own products in order to maintain their competitive advantage. The proliferation of computers and distributed processing has brought about a need for standardization amongst communication equipment vendors.

The Open Systems Interconnection (ISO-OSI) 7-layer [77, 105, 110, 176, 212] architecture was designed for *interoperability* [184] of computer systems in a *heterogenous* [183] environment. The layered OSI protocols are designed with as much generality as possible with upper layers making use of the services offered by the lower layers. The interconnection of multiple computer systems requires a high degree of cooperation between them. The term 'Open' [29] was chosen to emphasize the fact that by conforming to those international standards, a system will be open to all other systems obeying the same standards throughout the world. The basis of OSI standardization [88, 126] is a reference model for the coordination of standards development. Existing standards

will be placed in perspective within the overall reference model, which is organized into functional layers.

The OSI reference model (OSIRM) is not intended to serve as an implementation specification nor as a basis for appraising the conformance of actual implementations. Rather, the OSIRM provides a conceptual and functional framework that allows for independent research and implementation on developing standards for each layer. The OSIRM, defined by ISO 7498 [105], was completed and adopted by ISO in 1984.

2.2 OSI Architecture and layering

The OSIRM uses a structuring technique called *layering* [58, 22]. The communication functions are partitioned into a series of horizontal layers (see figure 1). Each layer performs a related subset of the communication functions. The *services* defined for each layer are performed by building on services provided by the layer directly below. Conversely, the services at each layer is called upon by those at the next higher layer. Thus when an application process initiates a communication, it passes its message down through each layer. One very important feature of OSI is that these service definitions are independent of each other. In other words, any service can be implemented at one layer regardless of the methods used to implement the services of the layers above or below it. This allows technological advances at any layer to progress without affecting the overall functionality of the communication stack, whereby each layer provides a standardized interface with its adjacent layers. The OSIRM is divided into seven layers.

The upper three layers provide the functions in direct support of the application process, whilst the lower three layers are concerned with the transmission of the information between the end-systems. The transport layer is the link between these two groups. It provides end-to-end integrity of the communications, ensuring that the appropriate quality-of-service (*QOS*) from the lower layers meet the requirements of the upper layers.

Figure 1 Open System Interconnection Reference Model

2.2.1 Why Layering?

The structured OSIRM was chosen as opposed to a monolithic one because communication between entities on different systems is too complex to be handled by a single unit. The most efficient way of solving a large problem is to break it into smaller sub-problems. However, too many layers would make the structure too complex, while too few would lead to very complex protocols. The seven-layer architecture was chosen not only to make the complex task of providing interconnection manageable, but also to provide an interface between the end-system and the transmission¹ process. Another view of a layered architecture is the onion skin analogy. Each skin of the onion represents a uniform layer of the architecture, and removing one layer reveals another layer which was previously completely concealed.

¹ The transmission process is responsible for transporting data packets between two communicating end-system processes via a physical communication path.

2.2.2 Layering Principles

Within the layered architecture, each layer must interact with the layer above and below. The upward interaction at the Application layer is with the user, and the downward interaction of the Physical layer is with the transmission medium. For each interaction, an interface must be defined. Typical examples of what must be specified includes the protocol and timing. The set of rules governing the exchange of data between two entities is referred to as a protocol. For two entities to successfully communicate, they must “speak the same language.” What is communicated, how it is communicated, and when it is communicated must conform to some mutually accepted set of rules between the entities involved.

The key elements of a protocol are:

- ☐ **Syntax:** Includes such things as data format, coding, and signal levels;
- ☐ **Semantics:** Includes control information for coordination and error handling, speed matching and sequencing;

The layer protocols operate on a peer-to-peer basis. Timing information is needed to detect failures so that maintenance procedures to correct faults can be invoked. The functions of each layer were selected to give maximum flexibility to implementors and to ensure that current widely used standards could be encompassed by the model.

Each layer can be viewed as a **logical grouping** of functions that provide specific services to the layer above. A function, or a group of functions, within a layer is called an entity. Interactions between entities from adjoining layers are supported by primitives which take the form of request, indications, responses and confirmations. In an interconnection, each layer must have at least one active entity. Each layer (N) has an upper layer (N+1) as an upper layer boundary and an (N-1) layer as a lower boundary as shown in figure 2. The upper boundary for the Application layer is with the application process, whilst the lower boundary for the Physical layer is with the transmission medium.

Figure 2 Layering Interface

Entities within the same layer communicate with each other via *peer protocols*, through the use of the (N-1) layer service, which provides a *logical connection* path.

Service access point (SAP) provides exchange of information between entities of adjoining layers. The lower (N-1) layer is referred to as the service provider as layer N receives services from it. Layer N is called the service user of layer N-1.

Peer-entities and entities in adjoining layers exchange a variety of information. They are broadly classified as follows:

- ☐ Interface Control Information (ICI)
- ☐ Protocol Control Information (PCI)
- ☐ User data (UD)

ICIs are transferred between adjoining layer entities to ensure that their cooperation across the layer proceeds smoothly. These are the service primitives. PCI is exchanged

Figure 3 Layer Interface

between peer entities to establish and maintain their joint operation. User data is the information which is transferred between peer-entities; its transfer is the justification for establishing the connection between these entities. The messages which are passed between entities are termed data units. User data is combined with PCI to form a protocol data unit (PDU), the data unit which is transferred between peer-entities. The PDU is combined with ICI to form the interface data unit (IDU), the data unit which is transferred across the interface between adjacent layer entities (see figure 3). When the IDU has been transferred across the interface, the ICI is no longer needed and hence can be stripped away to leave the service data unit (SDU).

2.3 Overview of the Seven Layers of the OSI Architecture

The services and protocol provided by each layer are briefly described as follows:

Application Layer — Provides the application process [78] with a window into the OSI environment. It contains *management functions* which are generally useful mechanisms to support distributed applications. Application processes perform many different func-

tions and need to use many different aspects of the OSI environment. This layer is not, therefore, as easily defined in terms of services and protocols as the other layers. Instead, a number of application types have been identified, corresponding with those general areas of application processing where OSI is particularly relevant. It has been further categorized into three service elements namely *common*, *application-specific* and *user-specific* service elements. The *common* service elements provide functions that are independent of the application. *Application-specific* elements provide the services needed by a certain category of application such as file transfer, access and management (FTAM) [10] and electronic mail (X.400) [203]. *User-specific* elements apply to functions that are unique to a certain application, for example, airline reservation systems.

Presentation Layer — Provides independence to the application processes from differences in data representation (*syntax*). It ensures that the information delivered is in a form that is understandable to the receiving system. This allows, for example, systems which use EBCDIC encoding to communicate transparently with other systems which use ASCII encoding. This layer may also provide data compression and encryption facilities.

Session Layer — Provides the *control structure* for communication between applications [70]; establishes and manages dialogues, and terminates connections (sessions) between cooperating applications.

Transport Layer — Provides *reliable*, transparent transfer of data between end points. This layer also includes functions such as multiplexing [73] and segmentation. The transport layer is the bridge between application-related functions (upper three layers) and transmission-related functions (lower three layers) and hence is the key to the operation of OSI. Its functionality depends largely upon the quality-of-service provided by the underlying network.

Network Layer — Provides upper layers with *independence* from the data transmission and switching technologies used to connect systems. Its main functions include routing and relaying through switched communications media.

Data Link Layer — Provides for the *reliable* transfer of information across the physical link. It sends blocks of data (frames) with the necessary synchronization, error and flow control procedures. Here, the flow control is point-to-point (between adjacent nodes) as opposed to end-to-end flow control provided by the transport layer.

Physical Layer — Concerned with *transmission* of unstructured bit stream over a physical medium. It deals with the mechanical, electrical, functional, and procedural characteristic needed to access the physical medium.

2.4 Why are protocol implementations slow?

The effective data bit rate delivered to the application program of an end-system depends largely upon the following:

☐ Hardware issues

- Host architecture and hardware it is executing on;
- Adapter architecture and hardware;
- Host-Adapter hardware Interface, e.g., DMA.

☐ Software and protocol structure issues

- Host operating system environment;
- Timer and Buffer management processing overhead;
- Implementation techniques used;
- The protocol itself, its PDU structure and its flow and error control mechanism;

2.4.1 Processing and data path of a typical application PDU

The implementation of a full protocol suite on an end-system is a complex and time consuming task. The choice of design depends largely upon maintainability, testability, performance requirements and its operating environment. The ISO Development Environment (ISODE) [143, 159] is an example of such product whose source is available to the public.

On send, data units are passed from a user process to the host operating system process through an application interface. In BSD UNIX, sockets [119] are used to provide bidirectional communications and queuing. Synchronization between the adapter and the host processor is either through interrupt or polling mechanism. Protocol processing on end-systems can be generally classified as follows:

- *User process* — Typically the application program and all upper layer protocols which do not reside in the host operating system program, for example, FTAM [16] application protocol and the presentation and session layers. It is the simplest to implement and maintain [193];
- *Host operating system program* — Communication software in the host operating system can be divided into the kernel program and device drivers [90]. The kernel program here is the communication software implemented in the system kernel and accessible via a set of standard system calls from the user process. Examples of this include TCP/IP and TP4/CLNP protocols. Device drivers are configured into the system kernel to independently cooperate with a particular external hardware device (network adapter). A typical example includes the Logical Link Control (LLC) layer which interacts with the local area network adapter that incorporate the Medium Access Control (MAC) and physical layers. The advantage of implementing a communications protocol in the kernel is that the software can take advantage of operating system services like scheduling, memory management and interprocessor communication. The software developed will be dependent upon the operating system environment and is generally not portable. Developing, maintaining and debugging software in the operating system kernel can be very tedious and time consuming.
- *The network interface adapter* — All end-system computers will have a network adapter to interface with the physical transmission medium. The most common of these include LAN adapters, such as FDDI [160, 3] and Ethernet adapter boards. The establishment of IEEE 802 LAN standards has encouraged the development of VLSI

Figure 4 Data Movement in End-Systems

chips which implement large parts of its defined protocols in silicon [34]. Some selected functions or the complete layer of the higher layer protocol functions may also be offloaded from the host to the adapter.

Figure 4 shows the data path of a typical application PDU as it moves from one end-system to its destination end-system.

Recent developments in high-speed LANs [2] and WANs [62], most notably the Asynchronous Transfer Mode (ATM) [19, 56, 154] are capable of delivering data rates close to the memory bandwidth of the host. The major problem is to deliver this bandwidth directly to the application program through a deeply layered standard protocol stack implemented on the end-system. This requires a well engineered software/hardware architecture to balance both the protocol processing requirements and movement of data.

2.4.2 Host Architecture

It is important to keep data movement on the workstations down to the minimum, as workstation architectures have been optimized for data processing rather than data movement [51]. RISC processors employ caches to bridge the gap between its Central Processing Unit (CPU) and main memory speeds. Cache reduces latency and also reduces contention for main memory access. Its effectiveness is however influenced by factors such as size, organization, locality and processor scheduling. For example, when an operating system switches context, the assumption of locality is violated because the instructions and data of the newly-scheduled process is probably no longer in the cache. Context switch thus has a greater cost than just the operations performed by the kernel. Advances in processor performance continue to outpace improvements in memory bandwidth [87], so the penalty for access from main memory due to cache misses will increase. As the bandwidth of high-speed networks approaches the memory bandwidth of modern computer systems, the cost of even a single data copy is significant. Hence, for bulk data transfer, the speed of the host memory and bus quickly become the bottleneck.

Data may be accessed from main memory for the following reasons:

1. Movement of data between kernel address space to user address space. This physical movement of data units is needed to cross memory protection boundaries [66].
2. Transfer between host memory and adapter buffer.
3. During Segmentation and Reassembly processing.
4. To separate data from a header of unpredictable length and to get data into alignment.
5. Per-octet processing may require data to be accessed physically — when data is not already in cache.

2.4.3 Adapter

One of the important issues in high-speed network research is the performance trade-off one makes between hardware and software implementation of protocols [146, 209]. In a hardware implementation, the hardware used in the adapter might range from

microprocessors to single-chip VLSI implementations. The choice of hardware depends largely upon the complexity of the functions supported by the adapter.

There is a range of network adapter design [155] from a performance perspective to support upper layer functions:

- The Minimalist approach where no upper layer protocol processing is performed on the adapter, and data movement is the responsibility to the host processor and software. For example the WITLESS [108] architecture proposed by Van Jacobson to support single copy implementations of TCP;
- Use of DMA to assist in the data movement activity while still doing very little processing on board [14];
- Perform time consuming checksum routines in hardware on the adapter [59];
- Perform the complete protocol processing on the adapter, thus off-loading the host CPU of almost all network I/O related activity [13, 28, 169].

2.4.4 Host/Adapter interface

It is often difficult to partition a protocol in a clear-cut way. So, the adapter may end up communicating with the host through a complex protocol which negates the advantage of offboard processing. Hence, it is also important to ensure that the communications load [12] between the adapter and the host is as small as possible. It should be much simpler than the protocol it supports. Some possible areas for improvement are:

- Minimization or elimination of host interrupts;
- Minimization of data copying;
- Minimization of host involvement in protocol processing;
- Minimization of host bus/memory usage.

There are several options available to transfer data packets from the host memory to the adapter [57, 60, 174]. They include:

- Direct Memory Access (DMA) between adapter and buffer managed by the operating system. The data must then be copied once again to the application address space. A given piece of data must travel over the system bus three times: adapter to memory, memory to CPU, and CPU back to memory.
- DMA between adapter and the application address space. Data only need to use the system bus once, from adapter to memory. One potential problem is that the transport protocol, which is implemented on the host software, may wish to defer data transfer to applications until header processing is complete.
- Programmed I/O where the host is involved in the transfer of data. Here, data will traverse the host bus at least twice: Once to get data onto cache and another to move it out.

2.4.5 Operating system support

Packet processing requires considerable support from the host operating system [47, 95]. For each incoming packet, the host processor must:

- Accept an interrupt;
- Allocate a buffer for an incoming packet;
- Copy data from network address space to host protocol processing address space;
- Schedule and Wake up one or more processes;
- Perform protocol processing of the protocol stack;
- Perform any timer management function;
- Copy data from host protocol processing address space to user process address space;
- Free buffer of processed packet;

The facilities offered by the operating system kernel in the host machine will affect the performance of the protocol implementation. Questions like “Does the operating system support concurrency?”, are important as communications software is concurrent in nature. For example, MS-DOS does not support concurrency, which may make it difficult to use for efficient communications software [7].

An expensive operation in communications processing is *context switching* [138]. Due to the high level of concurrency in communication software, explicit support for lightweight tasks in the kernel is generally essential for high performance protocol implementation. Lightweight processes (LWPs) are processes that can be efficiently created, destroyed, scheduled, and context switched. These LWPs commonly referred to as threads make asynchronous interprocess communications more attractive. Multiple LWPs usually share the same address space. This reduces the overhead for thread creation and context switches between threads. The x-kernel [94, 95, 149] was designed for constructing and composing network protocols. It supports multiple address spaces and lightweight processes. Its primary objective is to facilitate the implementation of efficient protocols. Not all operating systems provide explicit support for implementing network protocols. At the extreme, the distributed V-kernel [15, 40] was designed specifically for a particular set of protocols, whilst the Mach operating system [157] moves the responsibility for implementing protocols completely out of the kernel.

Every incoming packet at the host-network interface invokes the interrupt mechanism to signal for a service from the host processor. The interrupt mechanism here is best suited for asynchronous communications in packet-switched networks. The processing overhead for these *interrupt handling* routines can also cause significant performance degradation depending upon the implementation and host operating system [146]. Minimizing process scheduling overhead is also important to achieve high performance.

2.4.6 Buffer Management

The most complex support function is the task that manages the buffers which hold the data being processed. Buffer management [213] can easily grow in complexity to swamp the protocol itself [44].

Typically, SDUs and PDUs and their associated headers are stored in buffers which are divided into buffer pools. Good buffer management should correlate to the traffic presented by the applications using the transport service. Typically, in order to maximize

memory utilization, buffers are allocated on demand and freed by buffer chaining techniques. This technique is efficient when a large percentage of packets on the network are short packets. Variation on buffer size affects the following:

- Small sized buffers reduces the wastage space of partially filled buffers;
- Large sized buffers saves on buffer allocation and deallocation overhead

Each layer may have an input buffer pool and an output pool, shared with the adjacent layers. For example,

1. Layer N receives a signal from the lower layer that a packet has arrived.
2. Layer N allocates a buffer space and copies the data onto its input pool.
3. Layer N allocates a buffer in its output pool before processing
4. Protocol processing for layer N performed
5. PDU copied to output pool
6. Layer N signals the upper layer that a packet is to be passed
7. Deallocation of buffer space once data are passed to the next layer.

This technique requires copying within each layer and across adjacent layers and dynamic allocation and deallocation of buffer space. Since data copying is on a per-octet basis, the processing overhead needed for such operations are very high. The widely-used buffer cut-through technique (e.g. [205]) passes pointers and hence reduces buffer copy across layers.

2.4.7 Timer Management

Timers are indispensable tools in building reliable communication systems. A timer is a *failure detection* tool. Any time-out based action is a guess based on incomplete information, and as such is bound to be non-optimal [211]. The fundamental problem in using a timer is that a time-out does not tell precisely what went wrong (or even whether there is anything wrong!). Assumptions have to be made when attempting to recover from these unknown failure. In TCP, every packet is coupled to a timer. On

sending data, a retransmit timer is set to the estimated value of the Round Trip Time (RTT) between the two communicating entities [115]. A good estimate of the RTT is important as it prevents unnecessary retransmission or long blocking times. On receipt of an acknowledgment, the timer is cleared. The overhead of managing these timers can be very high. Everytime a clock interrupt occurs, all the timers in operation have to be decremented and checked for underflow. Some operating systems timer packages are not designed for such demanding use and hence a specialized timer package [199] may be used to increase its performance.

2.4.8 Software Implementation

The execution speed of a protocol stack may vary widely between implementations. Each layer entity may either be implemented as a process or as a procedure call. The layer processes interact via some synchronization mechanism [164]. Synchronous intraprocess communications like upcalls [49] may be used to transfer messages up and down the protocol stack. In asynchronous interprocess communications [206], techniques like message queues are used. Since message queues do not necessarily block the sender, it is possible to process messages in each protocol layer in parallel, potentially increasing overall throughput. However, this advantage may be offset by additional context switching and data copying overheads.

2.4.9 Protocol structure and mechanism

Protocols can generally be classified as heavyweight protocols and lightweight protocols [64]. Standard protocols like OSI TP4 and TCP fall under the first category. Lightweight protocols are designed to be "lean" and fast, with the emphasis on high speed execution either by hardware or by reducing protocol complexity and removing any unnecessary functionality. This generally restricts the protocol to support a particular type of service more efficiently whilst performing poorly for other services [151].

2.5 Software Performance Engineering Principles

The objective of this thesis is to achieve an optimal implementation for high performance. Certain software performance engineering (SPE) principles have been compiled by Smith [172, 171] to suggest improvement in software performance. They will help in constructing our proposed high-speed protocol processing architecture to meet our performance objectives (see section 4.2). These general principles are also used to compare current state-of-the-art techniques and our proposed approach. SPE principles do not replace performance models, but supplement them for engineering systems to meet performance goals. They can be used to avert perilous problems, but it needs quantitative analysis to determine their performance gains. To experienced performance analysts, these principles are not revolutionary new prescriptions. However, they are a generalization and abstraction of the "expert knowledge" that performance specialists use in building or tuning systems. The six performance principles are as follows:

- *Centering — Independent*
- *Processing versus Frequency trade-off — Independent*
- *Fixing-point — Independent*
- *Locality — Independent*
- *Parallel Processing — Synergistic*
- *Shared Resources — Synergistic*

They can be broadly classified into two categories: independent and synergistic principles. Independent principles deal with improving the tasks' own performance parameter. For example, reducing its resource requirements by applying the centering principle. Synergistic principles, on the other hand, improve the overall performance through the cooperation with other tasks, for example, parallel processing. It is sometimes difficult to determine performance trade-offs, particularly with synergistic principles, as they have many complex interrelationships. Modelling and evaluation can resolve these ambiguities [112]. Most present protocol implementations are more concerned with

correctness rather than performance. It is possible to tune an existing system to improve performance, but the gains are usually modest. Any dramatic improvement generally require a major modification to the software.

Centering Principle — The *Centering principle* identifies the dominant work load functions and minimizes their processing. Centering is based upon the “80–20” rule. For example, the execution of code within programs generally have less than 20 percent of a program’s code which accounts for more than 80 percent of its computer resource usage. Performance enhancement made to these key areas will greatly affect the overall performance of the system.

Processing versus Frequency trade-off Principle — The *Processing versus Frequency trade-off* principle minimizes the processing times frequency product. This suggests considering both the number of times software processing executes and the amount of processing per execution and minimize the product. It looks for trade-off opportunities in possibly performing more work with less frequency and vice versa. For example, it is desirable to maximize the use of overhead costs such as context switching.

Fixing-Point Principle — The *fixing-point principle* states that fixing should be done at the earliest possible time. The fixing “point” is a point in time. Fixing is often a trade-off between flexibility and response time. Early fixing leads to better performance, whilst later fixing may be more flexible and general. For example, a program can be either compiled or interpreted. Compiled programs fixes the machine instructions before run time, whilst interpreted programs chooses instructions at run time.

Locality Principle — The *locality principle* states that systems should be designed to have a close mapping of logical tasks to physical resources. For example, in operating systems, the term locality usually refers to memory-reference patterns. The locality principle can be applied for effective memory use such as cache usage.

Parallel Processing Principle — *Parallel processing principle* states that parallelism should be exploited when speedup is greater than the communications overhead imposed

by parallelism. For example, processing time can sometimes be reduced by partitioning a process into multiple concurrent processes which can then execute on different processors. Concurrency will be effective if the processing-time reduction is much greater than the additional overhead for communication and coordination of the concurrent processes. The performance improvement must also be weighed against the cost of the additional processing power and the cost of more complex software, to determine whether it will be effective.

Shared Resources Principle — The *shared resource principle* states that when exclusive access to a resource is needed, the sum of the holding time and scheduling time should be minimized. Computer resources are often limited; thus processes compete for their use. By sharing resources (rather than taking turns), the overhead for scheduling and the wait time to gain access must be minimized. For example, contention delays degrade performance; the intensity of the contention quantifies the effect.

This chapter introduced the protocol layering concept and general software performance engineering principles. It also examined why protocol implementations are slow. The next chapter reviews state-of-the-art techniques which are proposed to improve protocol processing performance.

Chapter 3

Review of the State of the Art in Performance Improvement of Upper Layer Protocols

3.1 Introduction

Network performance requires a variety of pieces to work together efficiently. On the hardware side the processor, the memory, the bus and the network adapter must all be fast, and must be able to communicate with each other efficiently. On the software side, the processing load in the processors is due to a combination of operating system overhead, protocol complexity, and per-octet processing on the data stream. The operating system overhead includes context switching [138], interrupt handling and process scheduling [26] associated with handling several concurrent stream of events. The rise in network signalling speeds has not been accompanied by equivalent gains in the bandwidth delivered to the application program [83, 185, 135, 182].

Recent work on high-speed protocol processing has explored various approaches (See figure 5):

- *Lightweight Transport Protocols;*
- *Special Implementation techniques;*
- *Hardware implementation;*
- *Parallel hardware implementation;*
- *Special Architecture or special protocol structures;*

- *High-speed presentation layer processing.*

These work can be categorized into two major research directions: The first approach focuses on improving the performance of standard protocols. It includes the ISO-OSI upper layer protocols and the Internet suite² [124]. They believe that most performance problems is due to poorly designed implementations of the protocols, and not the protocol itself. On the other hand, some researchers feel that these standard protocols are too complex to be efficiently implemented and hence take the alternative approach of redesigning newer protocols or protocol stacks that are "tailor-made" for high-speed networks.

The advantage of the former approach is that it can benefit from a large installed base whilst maintaining international standards. Standards typically take a long time to evolve and encompass social, political and economic issues. In this thesis, the first approach is taken. It is believed that minor extensions can be included into existing standard protocols to cater for high-speed network characteristics. For example, research into extensions of TP4 for high-speed networks are presented in [53, 200, 189, 69]. TP5 [91] is a transport protocol designed at INRIA (France) for the transport of real-time multimedia data, in addition to normal data. The protocol is a variant of TP4. Flow and error control are not performed on real-time data at the transport layer. A new type of PDU is added, namely RT (Real Time) TPDU which carries real time traffic. TP5 uses all the base mechanisms of ISO TP4 for normal data transmission, and is compatible with TP4. Flow control is replaced by rate control. On connection setup, the data rate is negotiated. Extensions to the session layer to support multimedia synchronization have also been proposed [156]

3.2 Lightweight Transport Protocols and their mechanisms to support high-speed networks

The complexity of error control and recovery mechanisms were brought about by the

² The Internet Suite is an industry defacto standard which is widely used today. It consists of essentially four layers: Network, Internet, Service protocols (transport service) and applications.

Figure 5 State of the Art Review

need to ensure reliable and error free transmission [152]. In “classical” protocols, e.g. TCP, the end-to-end flow and error control mechanisms are based on window flow control [166]. Error control is incorporated into the windowing mechanism by sending a positive acknowledgment on receipt of valid, uncorrupted and in-order packets to the sender when updating the window credits. With Optic fibres, the bit error rate has dropped from 10^{-6} in copper coaxial cables to 10^{-12} or better [181] whilst its transmission bandwidth has increased by a few orders of magnitude. Hence the main source of errors in optical transmission is packet loss due to overruns rather than corrupted packets. The main objectives in designing a lightweight transport protocol [161] are as follows:

- Optimize and tailor the control mechanism specifically for high-bandwidth, low-delay and low bit-error rate transmission;
- Minimize the amount of processing needed for data transmission and reception.

3.2.1 Summary of Transport Service Functions that affect performance of high-speed networks

The transport layer provides end-to-end flow control rather than point-to-point flow control as in the datalink layer. An end-to-end flow control is more complex especially if the underlying delivery mechanism is a connectionless type (datagram) which does not guarantee a reliable ordered delivery of data packets. The ability of the transport layer to detect, report, and correct errors determines the reliability of its transport service. The main features of a transport protocol are to provide a complete, in-sequence and error free delivery of the user data. The mechanisms required to support the transport layer service are:

- Connection Management
- Flow control and acknowledgment
- Error handling

3.2.1.1 Connection Management

Its functionality includes connection setup and release, selecting transport service, multiplexing and selecting packet formats:

- *Connection setup and release* can be broadly classified as:

- ☐ Handshaking Scheme or
- ☐ Implicit Scheme

In the *handshake* scheme, an explicit exchange of messages between communicating entities is required to open or close a connection. In an *implicit* scheme, a connection is setup on receipt of the first transport protocol data unit (TPDU) packet and released by a timer-based mechanism. The implicit scheme is very efficient for short term connections like remote procedure calls where small amounts of data are exchanged and low network delays are desired.

- *Negotiation of transport service* in a connection is essential since there maybe a *disparity* between the services provided by the network layer and the requested service from the upper layers. Here, the transport layer functionality can vary substantially. Furthermore, depending upon the environment, parameters like maximum packet length, timeout values and buffer size must be negotiated between the end systems. Parameter negotiation may occur during the setup phase and remain static throughout the connection or updated during the data transfer phase (dynamic).

- *Multiplexing* allows several connections to use the same set of lower-layer resources. These *savings*, however, are obtained at the *expense* of having to multiplex and demultiplex packets, and also to ensure fair sharing of resources.

- *Packet format* describes the layout of the PDU fields carrying control information and user data. The size and the alignment of control fields in a packet header has a significant impact on the speed of packet processing. *Word alignment* as opposed to multiple fields being packed into a single word can reduce processing overhead as it

reduces expensive masking and shifting operations. With high speed transmission, it is acceptable to sacrifice communication bandwidth by using word aligned fields to simplify encoding and decoding of protocol headers [163].

3.2.1.2 Flow Control and Acknowledgment

A *flow control* mechanism avoids network congestion. *Window flow* control [50] and *rate* control [25, 65, 72] are two methods of controlling the senders data flow. With window flow control, the receiver specifies the maximum amount of data, called a window, that the sender may transmit up to its upper window edge until the next window update. The window-based control integrates both a flow and error control mechanism. The retransmission mechanism used for error recovery includes stop-and-wait, go-back-N and selective-repeat [175, 166]. The selective-repeat mechanism is bandwidth efficient as it saves unnecessary retransmission, but requires a bigger buffer for both the sender and receiver. On the other hand, the stop-and-wait mechanism is the easiest to implement but suffers from very low throughput especially in high bandwidth-delay networks.

Rate control requires the sender to know the rate at which data can be transmitted and its burst size, which specifies the maximum amount of data that may be sent as a burst of packets. It matches the rate at which data are offered to the network by the sender, to the rate at which it can be transported and processed by the receiver. This is especially important in high speed networks to prevent overruns. The rate at which data is transmitted must be limited by the data rate through the transmission network, and by the rate at which the receiver can process the arriving packets. Window flow and rate control may be used together. Here, data are transmitted at a rate according to the rate control algorithm as long as there is enough window credit available.

Acknowledgment is used by the receiver to notify the sender of a successful or unsuccessful reception of data. Acknowledgments generated by the receiver in response to explicit or implicit events initiated by the sender is referred to as *sender-dependent* acknowledgments. Most protocols uses this mechanism. *Sender independent* acknowl-

edgments on the other hand are initiated asynchronously by the receiver, independent of the sender. This can be accomplished by a periodic timer interrupt mechanism. Here, the receiver need not worry about acknowledgment to the sender upon receipt of data and hence less processing is needed here.

3.2.1.3 Error Handling

Error handling is necessary to achieve reliable transmission between two entities. It includes *error detection*, *error reporting* and *error correction* mechanisms. *Error detection* is performed by means of sequence numbers and checksums [30]. Sequence numbers are able to detect missing data, out-of-sequence delivery and duplicates. Checksums detect bit errors in a data packet. In *error reporting* by the receiver, a negative acknowledgment (NACK) packet may be sent whenever an out-of-sequence packet is received. The retransmission mechanism is used for *error correction*. The methods used to trigger retransmissions are *Positive Acknowledgments with Retransmission* (PAR) and *Automatic Repeat Request* (ARQ). With PAR, the receiver acknowledges data received error free and in sequence. Retransmissions are triggered by timeouts at the sender when there is no acknowledgment from the receiver. With ARQ, retransmissions are invoked by the receiver which sends control information when errors are detected. Either Go-back-N or selective repeat mechanism can be incorporated.

3.2.2 Lightweight Transport Protocols

Lightweight transport protocols [64] are designed to be *fast* and “*lean*” and they differ essentially in the design of their mechanism used to support the transport layer service. The ISO-TP4 [104, 101] will be the basis of comparison between the various lightweight protocols surveyed here. Its protocol mechanism is almost identical to TCP, the transport protocol of the Internet suite [177, 137] which is now widely used. TP4 data delivery services roughly match TCP services. The reliable data delivery service is message-oriented, not stream-oriented as with TCP. TP4 is a connection oriented

window protocol which uses packet-based sequence numbers compared with a byte-based sequence numbering scheme used in TCP.

- *TP4* [54] uses a *handshake* scheme for connection setup and release. It allows static service selection such as maximum packet size during the connection setup phase and also dynamic update of service parameters during the data transfer phase, e.g. flow control parameters like window size. It uses the *sender-dependent* acknowledgment scheme and a sliding window to provide end-to-end flow control. TP4 uses the *PAR* mechanism to perform error correction. The choice of a timer value affects performance dramatically [115]. A small value may cause unnecessary retransmission, whilst a large one may cause performance degradation due to the under-utilization of the communication medium bandwidth. The packet format is not aligned in word boundaries for efficient implementation and the checksum parameter is placed in the packet header, which makes it difficult to incorporate hardware checksum calculations on-the-fly.

All lightweight protocols discussed here assume an unreliable (datagram) underlying network service providing best-effort delivery. They are described briefly below by highlighting on their key innovative features, and are arranged in chronological order.

- *Delta-t* [202] uses *timer-based connection management* to support lightweight connections. Connections are established *implicitly* on receipt of the first TPDU, and closed automatically after the last exchange of data. For a given stream, the sender and receiver maintain connection state and a send-timer and a receive timer respectively, when data are being exchanged. The rules applied to the receiver timer assure that the connection state is maintained long enough so that all old duplicates are detectable, assuring reliable connection opening, and that the node will not accept packets for a safe interval on crash recovery. The send-timer rules assure that sequence numbers will not be reused, even in the face of a node crash, until all data or acknowledgment packets with a given sequence number have expired. These rules are essential in maintaining a safe connection management in timer-based connection management systems, as packets "floating" in the

network due to a crash at the sender or receiver (loss of state) may fool the system into establishing another connection. One of the problems here, as with any timer-based system is determining the correct timer parameters for correct and optimal performance.

- *NETBLT* [46, 45, 122] was an early attempt to design transport protocols for high *throughput* bulk data transfer, by reducing the number of acknowledgments, and by the use of both rate and window flow control mechanisms. It reduces acknowledgment overhead by acknowledging whole buffers [33] rather than each packet. Window flow control is at the buffer level where the unit of transfers is in terms of buffers rather than packets. This reduces the number of interactions required between peer entities for data transfer. The sending and receiving clients confirm that they have set up matching buffers before any buffer can be transmitted. Either client can control the flow by not providing a new buffer. It can maintain a number of buffers at any one time to allow the sender to transmit new data whilst acknowledgment and error recovery for old buffers are taking place. Rate control is used to control the transmission rate of groups of packets. All packets are “longword-aligned”, i.e., all packets are multiples of 4-byte lengths.

- *VMTP* [38, 37, 39] was designed for *transaction* oriented type of processing, particularly remote procedure calls (RPCs), by using fast *implicit* connection establishment. It provides transport communication between entities via message transaction. The basic VMTP message transaction is: a client sends a request message to a server. The request will be terminated when the server respond to the request, much like a client-server model. It uses the interpacket gap to implement rate control by varying the time gaps between transmission of packets. It reduces the complexity of timer management by using only one timer for each client with an outstanding transaction, regardless of the number of packets in that transaction. Upon expiration of the timer, the client either retransmits the request or aborts the transaction with a failure indication.

- *XTP* [41, 153] combines the network and transport layer into a common transfer layer, and is designed to be implemented in *VLSI*. The XTP algorithms are designed to suit

the demands of the VLSI execution environment, the functional requirements, and their performance goals. It is designed to be decomposed into a number of parallel state machines. Most header fields are 32-bits wide with its checksum field located at the trailer.

Table 1 compares the major features of these newer lightweight protocols with TP4. We believe that research in efficient flow and error control schemes for high-speed networks is important to enable better utilization of the available transmission bandwidth.

3.3 Special Implementation Techniques

Special implementation techniques have been designed for protocol processing as its performance is highly dependent upon its **implementation** [150, 158, 167, 183]. Studies by Watson and Mamrak [201] have shown that up to 80% of communications processing is **not protocol related**. These include interlayer processing overhead and interrupt handling. They also described several means of gaining efficiency by appropriate design and implementation choices. A communication protocol like TP4 is not a piece of software, but it specifies the format of the data and acknowledgments that two end systems exchange information to achieve a reliable transfer. It does not dictate the details of the interface between an application process and TP4. This allows for flexibility as the implementors use the host operating system [119], and whatever interface it can offer.

Clark suggested using upcalls [49], where the processing of an incoming or outgoing message through multiple layers is handled, as far as possible, within a single process. The interlayer communication takes the form of procedure calls. There are two types of interlayer calls: an upcall, which is a call from the (N)-entity to the (N+1) entity, and downcalls from the (N+1) entity to the (N) entity. Woodside [207] studied various software communication architectures for parallel protocol execution with synchronous interprocess communication (IPC). In most operating systems, IPC represents significantly higher overhead than procedure calls [144].

<i>FEATURES</i>	<i>TP4</i>	<i>Delta-t</i>	<i>NETBLT</i>	<i>VMTP</i>	<i>XTP</i>
Connection Setup and Release	Handshake	Timer-based Implicit Setup and Release	Handshake	Implicit Setup and Release	Implicit (Setup) and Explicit Handshake Release
Multiplexing	Allowed	Allowed	Allowed	Allowed	Allowed
Packet Format	Minimal Alignment	Minimal Alignment	Longword Aligned	Byte aligned	Longword Aligned
Acknowledgment	Sender Dependent (on reception of data)	Sender Dependent (on reception of data)	Sender Dependent (on reception of data)	Sender Dependent (on reception of data)	Sender Dependent (explicit call by sender)
Flow Control Mechanisms	Window based	Window based	Window and Rate flow control	Window and Rate flow control	Window and rate flow control
Error Detection	Sequence Number and Timers	Sequence Number and Timers	Sequence Number and Timers (Receive side)	Sequence Number and Timer (Only one timer used)	Sequence Number and Timers
Error Reporting	None	Negative Acknowledgment (Optional)	Selective Reject	Selective Reject	Selective Reject
Error Correction	Positive Acknowledgment with Retransmission (PAR)	Automatic Repeat Request (ARQ) with Go-back-N retransmission	ARQ with Selective Repeat retransmission	ARQ with Selective Repeat retransmission	ARQ with Go-back-N or Selective repeat (Optional) retransmission
Sequence numbering scheme	Packet based	Byte based	Packet based	Packet based	Byte based
Checksums	Entire TPDU (Optional)	Header with data (Optional)	Header with data (Optional)	Header with data (Optional)	Header with data (Optional)

Table 1 Comparison of Lightweight Protocols

Clark [44] also argued that classical protocols like TCP (without checksum), if implemented correctly, can achieve throughputs of up to 800 Mbps on a standard 10 MIPS RISC-based processor. His techniques focus on reducing overhead such as:

- ☐ Reducing data movement inside the processing node [205];
- ☐ Use of suitable internal queue and buffer management strategies;
- ☐ Low overhead process structure (context switching);

Techniques such as remapping of pages associated with the application's virtual address to the operating system virtual address [59, 66] are proposed to minimize data movement on the host. A hybrid technique called fbufs [67] combines page remapping with shared virtual memory. Shared virtual memory employs buffers that are statically shared among two or more domains to avoid data transfers. It is possible to move application PDUs from the application address space directly to the adapter buffer via DMA provided no per-octet processing is required by the host. This is the ideal situation as data packets from the host memory are being accessed only once and the host is not involved in this operation.

3.3.1 Header Prediction

The header prediction algorithm was proposed by Jacobson for the receiving end of TCP [107]. The idea is to precompute what values should be found in the next incoming packet header for the connection, i.e. the receiver has 'hints' about what the next packet would look like. If the packets arrive in order, a few simple comparisons will suffice to complete header processing, otherwise it flows through the normal processing path. He eliminated data copies in the internal TCP buffers, modified the Sun ethernet driver to allow more than one packet to be queued for transmission and implemented the header prediction algorithm. He achieved 8 Mbps between two SUN 3/60s, compared to 3.5 Mbps in the original implementation of TCP/IP over Ethernet [4]. The header prediction algorithm improved the performance of the protocol processing part of TCP, but its impact on the overall performance is not as impressive when TCP is used in conjunction with other protocols to form a functional stack. It does not reduce non-protocol related processing at layer boundaries. Hence, one of the conclusions that Jacobson made was that "the prediction didn't make much difference in overall performance — it just made a small part of the work smaller" [109]. This observation is consistent with the conclusions arrived by Watson [201]. The bypass concept used in this thesis is a *generalization* of the

header prediction algorithm to multiple layers and hence addresses the other significant performance bottleneck of interlayer processing overhead as well.

3.4 Hardware Implementation

Since current reliable transport protocols are designed with software implementation in mind, they are relatively complex and have many different packet formats. VLSI implementations so far proposed have almost exclusively relied on “made for VLSI” protocols [11, 165, 113]. The XTP protocol was designed as a lightweight protocol with ease of VLSI implementation in mind. In essence, XTP is not the simplest protocol, nor is it the most complex. It is a minmax compromise: the most useful set of functions consistent with a simplified implementation. XTP is still too complex to be implemented entirely onto VLSI and hence its current design uses four custom made VLSI circuits [181] to provide a high speed data path for network traffic and a general purpose RISC processor to support the more complex state and interfacing processing. The long term goal for XTP is to represent the complete design in VLSI. The PROMPT [11] protocol was also designed to ease VLSI implementations that take advantage of its intrinsic parallelism. Rather than implement one or more upper layer protocols on the network adapter, the x-chip [9] implements a collection of protocol-independent functions, like multiplexing, on the network adapter.

One way to increase performance is to implement the entire functional stack in hardware [131, 168]. *Hardware implementation* such as the VMP Network Adapter Board [114], was designed specifically for the lightweight protocol VMTP. The Nectar Communication Accelerator Board (CAB) [6] can be programmed to work with various protocols. The CAB communicates with the host memory directly, and its programmability can conceivably be used by applications to customize the protocol processing stack. Cooper *et al* [57] report that TCP/IP and a number of Nectar-specific protocols have been implemented on the CAB (connected to Sun-4 processors).

Hardware assist of the full TP4 protocol logic using general purpose processors was proposed in [169]. It only achieved marginal gains. In UltraNet [13], the TP4 checksum field was modified to reside on the trailer to ease hardware implementation. All other TP4 functions are executed on a RISC processor. The Hewlett Packard Express Card [28] is an attempt to place the entire OSI layers 1–6 on an adapter board.

3.5 Parallel Hardware Implementation

- *Parallel hardware implementation* techniques have also been studied by Zitterbart [214, 215, 216], Giarrizzo *et al* [81] and by Jain, Schwartz and Bashkow [111] to speed up conventional transport protocols like TP4.

Zitterbart applies the concept of vertical and horizontal subdivision structuring to layered protocols, in particular the transport and network layers. She proposes a send and receive pipeline (structural parallelism) with the necessary interactions which she terms events, as these two pipelines are not completely independent. For example, on receiving a packet at the transport layer, an acknowledgment packet may be generated and placed in the send pipeline.

She also explored opportunities for parallelism inside the pipeline stages. She categorizes functions within a protocol layer according to whether there are any data dependencies between them and whether they are synchronous or asynchronous within the layer. Data dependent functions refer to protocol functions whose output data are necessary input data for another function. Synchronous functions refer to functions that occur synchronously to the flow of packets through the layers. An asynchronous function is one which is invoked asynchronously in the layer entity, e.g., timers.

Using this classification, three basic building blocks — pipeline, array and irregular blocks — are used to configure the protocol machine for further parallelism (functional parallelism). For example, pipeline blocks are used for synchronous functions that are data dependent and array blocks for synchronous data-independent functions. Figure

Figure 6 Zitterbart: Structural and Functional Parallelism

6 illustrates the structural and functional parallelism proposed by Zitterbart and her classification of the building blocks.

Giarrizzo *et al* discuss [81] several possibilities for parallel processing within the transport and data-link layers. They view a protocol as a composition of state machines which can be implemented on different processors. Thus the behavior of most protocols is specified in the form of extended finite-state machines or state-transition tables, which apply to one (logical or physical) connection. For each connection served by a protocol layer, at least one state machine exists on each side of the connection. Conceptually, each state machine may be implemented as a cyclic process, which runs more or less independently. Synchronization mechanisms are required when access to shared resources (e.g. timers or buffers) are invoked. Both Zitterbart and Giarrizzo make use of the

Figure 7 Jain Schwartz and Bashkow: Packet level Parallel processing of TP4

transputer, a commercially available parallel microprocessor, as the basic building block for their implementation.

Speed up with functional parallelism is limited by the number of functions (tasks) that packet processing represents. A pipeline architecture is limited by the unequal processing requirements of each layer. For example, the processing requirements of the session and transport layers are not equal in the data transfer phase (see section 5.7). Another form of parallelism for protocol processing is packet-level parallelism. Jain, Schwartz and Bashkow [111] proposed a packet level parallel processing architecture consisting of 14 Motorola 68030 processors to handle TP4 processing at 1 Gbps rate. Figure 7 illustrates their proposal.

The queue processor has two primary functions. It sets up and maintains ACK queues which are needed to ensure an ordered delivery of arriving packets and the generation of ACK packets. Each ACK queue structure consists of a set of pointers to the processed packets in the multi-processor pool and their corresponding sequence numbers. This will enable that in-order and error-free packets can be passed to the next layer. Each logical connection (multiplexing) has its own context record which is accessible by the ACK queue processor. They assumed that the checksum field is located at the end of the TPDU packet (not true for both TP4 and TCP) so that checksums can be computed on the fly by the input and output processors.

Bjorkman and Gunningberg [21] studied the effects of memory contention for shared state information in a multiprocessor TCP/IP implementation and concluded that mutual exclusion by locking can be costly if the parallel protocol code frequently access shared state and data. This effect was also confirmed by Goldberg *et al* [82] and Ito *et al* [106] who considered the full OSI stack.

Other attempts in parallel TCP/IP implementation are described in [162, 132, 120]. Parallel implementation of the XTP [32, 63] protocol using transputers were also proposed.

3.6 Special Architecture or Special Protocol Structures

This section describes four proposed solutions that support very high speed communications by reducing the number of layers in the full protocol stack. Clark and Tennenhouse proposed Integrated Layer Processing (ILP) [43], Sterbenz and Parulkar [179, 180, 147] proposed a new host-to-network architecture (AXON), Haas [84, 85] proposed a Horizontally Oriented Protocol Structure (HOPS) and Zitterbart *et al* proposed a flexible high-performance communication subsystem [218].

3.6.1 Integrated Layer Processing

Clark and Tennenhouse [43] proposed Integrated Layer Processing (ILP) to minimize per-octet processing overhead. Unfortunately, traditional deeply layered protocol stacks often impose "precedence" or "ordering" constraints that limit opportunities for optimization. Each layer usually executes its protocol functions according to an internally specified sequential FSM. Hence, sequential processing is implied for the individual layers.

Another major problem with ILP is that data manipulation performed by program modules in different protection domains cannot be easily integrated. The motivation for ILP is that the load and store operations, to/from external memory and cache, required by data manipulations are expensive and should be minimized. While caches are very effective in reducing memory accesses for many computations, the characteristics of strictly layered message processing are such that caching is not as effective. Furthermore, there is still a cost for accessing the cache. In the strict-layered for-loops, each time a word of data is manipulated, it is loaded and stored. In contrast, each word is loaded and stored only once in the integrated for-loop, even though it is manipulated more than once. In order to achieve ILP, they also proposed Application Level Framing to provide a single common unit of data which they termed ADU.

3.6.2 AXON

The goal of the AXON [179] project was to design a front end communications processor with the ability to perform protocol functions in real time with no packet buffering and to implement it in VLSI. The main features of the Axon architecture are:

- ☐ An integrated design of host and network interface, operating systems and communications protocol. The Axon host architecture and network interface is designed to provide a path directly between the network and host memory, without any need for data copy.
- ☐ A network virtual storage facility which includes support for virtual shared memory on loosely coupled systems.

- ① Packets arrive at the receiver
- ② Communications Processor decodes header and indicate to host that a packet is present.
- ③ The Host system determines the address of the packets destination and sends the page addresses to the Communications processor
- ④ Writes packet to memory via DMA

Figure 8 Sterbenz and Parulkar: AXON Block Diagram

- ❑ A high performance lightweight transport protocol.
- ❑ Communications processor with a pipeline architecture of very fine granularity (bit/byte operations). Incoming packets are initially channelled through a pipeline of processing modules, whereby control information is separated from the data stream. The control stream then flows through another pipeline (byte operations), where the header information is extracted and decoded. The destination address of the data is determined here so that the processed data stream can be copied directly into the host address space. On the other hand, the data stream is passed through a pipeline (bit operations) of protocol modules like presentation encryption.

Figure 8 shows the processing path of arriving packets.

Application
Presentation
Session
Transport
Network
Data-Link
Physical

■ Functions like Flow Control, Session Management and Presentation conversion

Figure 9 Haas: HOPS Architecture

3.6.3 HOPS

In the HOPS architecture proposed by Haas, protocols are divided into functions instead of layers. His proposed architecture is divided into 3 "layers": the Network Access Control (NAC) layer, the Communication Interface (CI) layer and the Application layer. The NAC layer consists of all services defined in the Network layer and below of the OSIRM. The CI layer merges the functionality of the Transport, Session and Presentation layers. The Application layer correspond to the application layer of OSI. Figure 9 illustrates the HOPS architecture and the functional block diagram of the CI layer.

The horizontal structure of the CI layer, coupled with a small number of layers, results in the elimination of unnecessary replication of functions and a reduction in interlayer processing.

3.6.4 Flexible High-Performance Communication Subsystem

The goal of the high-performance communication subsystem proposed by Zitterbart [218] is to obtain a flexible and yet efficient protocol implementation whose protocol

functions can be individually configured for a particular application. Protocol machines are created, by combining a minimum set of protocol functions needed to support the quality of service requirement for the application, using a protocol configurator. These functions are grouped according to the basic building blocks proposed earlier by Zitterbart (see section 3.5) and can be mapped onto a multiprocessor system.

3.7 High-Speed Presentation Layer

Most of the current research in high-speed upper-layer protocol processing discussed so far is concentrated on the transport layer. The presentation layer [61, 86] which demands very high processing power for presentation conversion is another major source of potential bottleneck [43, 83], because it touches every octet in the data. Thus it is important to optimize the presentation conversion procedures especially for bulk data transfer. One may consider tuning the implementation of existing standards, or provide alternative protocols such as the lightweight presentation protocol proposed by Huitema [92, 93]. Bilgic *et al* [20] proposed an ASN.1 encoder/decoder system architecture implementable by using VLSI hardware. It divides the complex nested encodings [178] into parallelizable pieces.

The next chapter characterizes the processing requirement for an end-system to support 1 Gbps rate. It will outline the goals of this thesis and the proposed solution. Classification of current state-of-the-art techniques in the context of software performance engineering principles will also be presented.

Chapter 4

Problem Statement and Proposed Solution

In order to deliver the network transmission bandwidth to the application program, all the intermediate processes along the path taken by the data units (DUs) must be able to support this rate. It includes both the processing requirements and any data movement across all layers including any physical boundaries, e.g., between end systems and adapter boards. Otherwise, the throughput will be limited by that of the slowest process.

4.1 Characterization of Gbps processing requirement

This section develops an equation to characterize the protocol processing requirement to achieve 1 Gbps rate.

Let,

X = average number of machine cycles required per octet (*cycles/octet*),

B = average number of machine cycles required per packet (*cycles/packet*),

P = packet size (*octets*), and

C = processor cycle time (*nsecs/cycle*)

The time taken to process a packet, T , is given by,

$$T = C.(X.P + B) \quad (4.1.1)$$

Hence, the throughput bound imposed by protocol processing, λ_{max} in bits per second, is given by the equation:

$$\lambda_{max} = \frac{8.P}{C.(X.P + B)} \quad (4.1.2)$$

Procedures are characterized into per-octet, per-packet or per-group-of-packets operation. The checksum and presentation conversion procedures are classified as per-octet operation (parameter X) as they operate on every octet in the message. These operations are very time consuming as they require a read operation from memory, protocol processing, and followed possibly by a write operation to memory. Per-packet procedures (parameter B) like header decode, which includes address decoding and multiplexing, are executed for each PDU. Per-group-of-packets operation is used to describe procedures like the transmission of acknowledgment (ACK) packets which is generated for each group of received PDUs. The ratio of PDUs received to ACK packets vary with implementation. These per-group-of-packets procedure timings are aggregated over the connection for all transmitted PDUs, and is included in parameter B .

Figure 10 shows the effect of varying X and B for a processor with a cycle time of $C = 40 \text{ nsecs}$. The effect of B on throughput performance can be reduced by increasing the packet size P . Whereas, a small increase in X reduces throughput significantly for large packets.

To achieve a 1 Gbps throughput rate,

$$\frac{8.P}{C.(X.P + B)} \geq 10^9 \text{ bits/sec}$$

$$\text{i.e.} \quad C \leq \frac{8.P}{X.P + B} \text{ nsecs/cycle} \quad (4.1.3)$$

If the it performs no per-octet operations, i.e. $X = 0$, with $B = 200$ machine cycles and $P = 1024$ octets, we require a processor with a cycle time of $C \leq 40.96 \text{ nsecs/cycle}$. RISC [18] based processors typically execute one instruction in one machine cycle provided memory can be fetched fast enough onto cache and the instruction pipeline is

Figure 10 Effect on throughput by varying X and B

not broken due to branch conflicts or context switches. Hence these values are achievable with current RISC processors (A RISC processor rated at a peak performance of 24.4 MIPS with $C = 40.96 \text{ nsecs/cycle}$). If P is increased to 8192 octets, a much slower processor with $C \leq 327.68 \text{ nsecs/cycle}$ is required.

If we now assume that $B = 0$, $P = 1024$ octets, and if X requires only 1 machine cycle per octet, then we need a processor with $C \leq 8 \text{ nsecs/cycle}$. Thus in order to process at 1 Gbps rate with $X = 1$, $B = 200$ and $P = 1024$ octets, the bottleneck processor require the equivalence of a processor with a cycle time C of 6.69 nsecs/cycle . This implies that a 150 MHz host processor³ is needed and that it is dedicated totally to communication processing, leaving no processor cycles for other host application programs. If the host processor is used for data copying or per-octet operations where the data is not in cache, the number of machine cycles (wait states) needed to access the external memory will be

³ Equivalent to a RISC processor rated at a peak performance of 150 MIPS

Figure 11 Processor requirement for 1 Gbps protocol processing

determined by the memory cycle time. As advances in processor performance continue to outpace improvements in memory bandwidth, the number of host machine cycles needed to access its main memory will increase, thereby increasing X . Hence, the throughput performance of per-octet operations do not scale well with any projected improvement in processor performance [87].

Figure 11 shows the effect of varying X and B on the processor requirement to achieve a 1 Gbps rate.

The above analysis assumes no segmentation/reassembly [117]. It also assumes no error and no retransmission. The actual throughput achievable will also depend upon the efficiency of the error control and recovery mechanism of the transport protocol used [166].

Table 2 lists the common protocol processing procedures and some major non-protocol related functions executed by the presentation, session and transport layers

during the data transfer phase.

The presentation layer could easily dominate the processing time for bulk data transfer, thus reducing throughput significantly as they are mainly per-octet type of operations. The session layer on the other hand will have little effect on throughput performance as synchronization and token management are invoked only periodically. For example in very large data file transfers, synchronization points may be inserted periodically for resynchronization in the event of failure in the network. For large data packets, checksum processing may dominate transport layer protocol processing.

4.2 Goals of this thesis

The simple analysis in the previous section shows that the throughput for bulk data

Layer	Procedure	Per-Octet (X)	Per-Packet (B)	Per-Group-Of-Packets Aggregated to Per-Packet for bulk data transfer (B)
Presentation	Encoding	X		
	Encryption	X		
	Compression	X		
	Context Alteration			X
Session	Synchronization Management			X
	Token management			X
Transport (Class 4)	Checksum (Optional)	X		
	Timer Management		X	
	Data TPDU numbering		X	
	ACK processing			X
	Resequencing		X	
All 3 layers	Header Construction		X	
	Header Decode		X	
	Buffer Management		X	
	Context Switching			X
	Data Copying using pointers within node		X	
	Data Copying across boundaries	X		

Table 2 Common procedures for Presentation, Session and Transport layers during data transfer phase

transfer is dominated by per-octet processing and data movement (X), since its cost increases proportionately to the size of the packet. On the other hand, per packet and per-group of packet processing overhead (B) can be reduced by increasing the packet length (P). It is anticipated that the proportion of bulk data transfer will increase on gigabit networks. Thus, one would expect to see more large packets (carrying bulk traffic) on the network in the future than we do currently [145].

The goals of this thesis are as follows:

- To study the impact of Gbps networks on throughput performance of a deeply-layered standard upper layer protocol stack on an end-system, and to alleviate any potential software bottlenecks using Software Performance Engineering techniques. These techniques are discussed in section 2.5.
- To deliver the raw network transmission bandwidth, up to the host memory bandwidth, to the application program. A 1 Gbps throughput translates to a host memory cycle time of 32 *nsecs* for a 32 bit bus. This is possible with current page mode DRAMs or static RAMs [55].
- To verify and extend the bypass concept in order to efficiently offload critical protocol functions to an adapter. Future trend in workstation technology favors using intelligent⁴ adapters even for workstations. Microprocessor technology is pushing towards lower instruction cycle times, making host bus and memory critical resources. Hence, any effort in reducing these critical resources will improve not only the communication performance, but also the performance of other application programs that are executing on the host processor. This argues for some form of hardware assist. A large performance gain can be achieved if we can offload all per-octet operations from the host leaving only a DMA transfer between the host main memory and the adapter.

⁴ Intelligent adapters are used to denote adapters that process some upper layer protocol functions [114].

- To enable Integrated Layer Processing of a standard multiple-layer stack. This work focuses on reducing non-protocol related functions like data movement, host interrupts and interlayer processing overhead which dominate total protocol processing time. These functions are typically handled inefficiently by current workstation architectures and operating systems.

4.3 State-of-the-art techniques from the SPE point of view

We should be able to apply SPE principles to protocol software. To begin with, we will show that the principles are already reflected in current state-of-the-art techniques for improving end-system protocol processing performance. These state-of-the-art techniques, already described in chapter 3, can be categorized by the SPE principle that best describes their main objective.

They are summarized as follows:

- *Lightweight transport protocols* — Lightweight protocols [64] can generally be categorized as following the *fixing principle* as their packet formats typically have fixed-length fields and are aligned on word boundaries. They also concentrate on reducing the complexity of protocols and tailor it to a specific set of applications. The main disadvantage is that they consider only a single layer. Hence most interlayer processing overhead still remains.
- *Special software optimization techniques* — Header prediction [107] is an example of the *Processing versus frequency tradeoff principle*. It predicts that the next packet arriving is from the same connection as the previous packet and that it is in sequence. If the prediction is correct, which it is most of the time, the processing is simplified. Otherwise a penalty is paid to reverse the action. It is normally the case that statistically, the packet currently received belongs to the same message transaction

to which the previous packet belongs [44]. The disadvantage is that this technique also attacks a single layer only and is applicable only to the receive path.

- *Special Architecture* — Specialized architectures like HOPS [84] follow the *locality principle*. Although they consider multiple layers, they require a complete redesign of the standard protocol stack. Another disadvantage is that each protocol stack variant has its own fully customized implementation and hence they are not flexible.
- *Hardware Implementation* — Upper layer protocol functions that are offloaded to an adapter which only consist of a single processor [13] can be categorized as following the *parallel processing principle*, since they share the upper layer processing load with the host. The efficiency of parallel processing techniques depend largely upon the considerations given to the *shared resource principle*.
- *Parallel Hardware Implementation* — The problem of shared resource contention is more acute in an adapter which uses more than one processor to support upper layer processing. Here, a pool of processors can be organized to perform protocol processing on a processor-per-packet, processor-per-function, processor-per-connection or processor-per-protocol basis. In the processor-per-packet model, an incoming or outgoing packet is processed on the next available processor and several processors may execute in parallel on different packets belonging to the same connection. The main disadvantage is that the connection state information has to be shared among the processors in order to be able to process packets belonging to the same connection in parallel [21, 82, 106]. Synchronization of accesses to the shared state information may involve mutually exclusive code regions or critical data protected by some locking mechanism. The problem may be worse in the fine grained processor-per-function [81] model since each processor performs a specific function which may require access to the shared data. On the other hand, both the processor-per-connection and processor-per-protocol model do not suffer from the same shared resource problem as the other two models, but their performance is limited. For example, in

the processor-per-connection model, each connection is dedicated to one processor. Hence the maximum performance gain for a single connection is equivalent to that of a single processor hardware implementation, and is not scalable. In the processor-per-protocol model, each processor is responsible for executing a separate protocol. Packets are passed from processor to processor on their way up or down the protocol stack. The disadvantage of this model is that packet processing will be pipelined at the speed of the slowest process, and the granularity for parallel execution is often too small for protocols with short execution times.

- *Lightweight presentation protocols* — It also follows the *fixing principle* and also address a single layer and hence have similar problems [17] as lightweight transport protocols. On the other hand, for bulk data transfer, any reduction in per-octet computation load is going to improve performance more significantly.

4.4 Characteristics of the Proposed Solution

This thesis uses a combination of approaches based on general Software Performance Engineering (SPE) principles to achieve our goals. The first part of this section develops the common desirable characteristics of the proposed solutions.

The *protocol bypass* concept employed is based on the *Centering* principle. Bypassing identifies a fast path for bulk data transfer, by which frequent simple operations are found and optimized.

The *Processing versus Frequency trade-off principle* is used to minimize the processing requirement of frequently executed functions. For bulk data transfer, where many data PDUs are transferred in a connection, protocol processing of these data PDUs dominate all other processing. This is especially true for large data packets with a high per-octet processing load. Only a small penalty is incurred if it fails the bypass test conditions (see chapter 5).

Since bypassing can be set up to identify a particular type of bulk data to be bypassed, depending upon the test conditions, the type and PDU format can be predefined and fixed. This is an example of the *fixing-point principle*. It reduces the code complexity and improve the execution speed of the bypass path.

The *locality principle* applied to multiple layer bypass, described in section 5.3, merges⁵ the protocol processing of adjacent layers thereby reducing interlayer overhead associated with context switching, local data copies and complex buffer management. This is similar to Integrated Layer Processing (ILP) (See section 3.6.1) proposed by Clark and Annenhouse [43] which requires a total change in the structure of standard upper-layer protocols. The layering architecture of the OSI model which includes complex connection establishment and service negotiation procedures at each layer, makes it difficult to combine these layers.

Chapter 5 describes a software solution which has the above characteristics. Using these common characteristics, we then evaluated the performance benefits of offloading protocol functions of the bypass path to an adapter. This is an example of the *Parallel processing principle*. It is important to consider the application of both the *parallel processing principle* and the *shared resource principle* together as the efficiency [68] of a parallel system is determined predominantly by the amount of time spent waiting for access to shared resources and by the amount of communication overhead. Two approaches are evaluated: The first approach implements the bypass path in VLSI (See chapter 6). The design applies the *shared resource principle* by minimizing the load on the host bus and memory, and reducing the number of interactions between the host and the adapter. In the second approach, a scalable packet level parallel architecture similar to Jain, Schwartz and Bashkow [111], but with the advantage that the complexity of the parallel system is drastically reduced and that a multiple-layer stack is considered (See chapter 7). A loosely coupled system is chosen where each processor operates

⁵ The OSI model does not apply any constraint on the implementation of a multiple layer functional stack, although reusability of codes for each layer should be considered in an implementation of such complexity.

on a separate local memory area, thereby reducing memory contention (*shared resource principle*). These two approaches maintain flexibility because they execute the more complicated (and infrequent) tasks in host software and achieves performance with hardware support for fast execution of the common functions.

4.5 Proposed Optimistic Principle

In addition to the six SPE principles described by Smith, [171, 173] we introduce the *optimistic principle*. This principle can be identified behind the basic notion of lightweight protocols that exploits the infrequency of errors in optic transmission, and also in Jacobson's [44, 107] header prediction TCP/IP optimization. A statistical technique such as optimistic prediction of host memory location, as proposed by Carter and Zwaenepoel [36] succeeds in avoiding user-kernel move typically 50% of the time. When the prediction fails, one requires additional processing and data movement to correct the mistake. They are based on the probability that arriving data packets in the data transfer phase are closely related to each other and that they are error free. Zitterbart [217] identified functions that are data independent and hence can be computed concurrently. For example, the routing function and header decode can be computed in parallel. In the event that an error occurs — a low probability in low error rate networks — the result of the routing function is just discarded. The Optimistic Principle can be stated as follows:

Execute optimistically only when the gain in delay offsets the cost of recovery weighted by the probability of failure.

The *Optimistic principle* can be generalized to any application that is “success-oriented”, i.e. the processing of common successful cases are minimized whilst a larger than average processing penalty may be tolerated to handle the rare occasion when the prediction fails. Hence on the average, if failure rates are low, the overall performance will increase. This notion can be extremely powerful in systems which have predictable behavior. The

optimistic principle can be categorized as an *Independent* principle as it deals with the tasks' own performance.

The *optimistic* approach is used to achieve greater concurrency and processor utilization in POBA (see chapter 7), where procedures like presentation decoding can be concurrently processed with protocol validation procedures. If the data is eventually found to be invalid, it is simply discarded and the effort used for decoding is "wasted". A suggestion for future work is presented in section 8.3 to improve performance by predicting that the connection establishment phase will be completed successfully and according to the requested service, so that data packets can be packetized and ready to be sent whilst waiting for the connection confirm primitive.

Chapter 5

The Bypass Concept and Software bypass

5.1 Bypass Concept

A bypass⁶, in the sense used here incorporates the performance benefits of a light-weight protocol by identifying a fast and “lean” path during the data transfer phase, whilst maintaining full conformance to the standard protocol. Figure 12 shows a Time Sequence Diagram (TSD) of the OSI Presentation, Session and Transport service layers for a connection. In the figure it is clear that, for bulk data transfer, the total time spent per connection is dominated by the data transfer phase. Hence throughput performance for bulk data transfer can be improved significantly by optimizing the processing path of the data transfer phase.

Consider just the transport layer, to see how a bypass does this: The full protocol implementation includes functionality for connection establishment, reliable data transfer, connection release and exception handling like network resets. Figure 13 shows the Finite State Machine (FSM) for OSI transport protocol classes 0 and 2. The full protocol implementation here has 7 states, 6 of which are related to connection establishment and release. Once the connection has been established, the system remains in the OPEN state during the data transfer phase. A transition to other states occurs only during a network reset (NRSTind) or explicit connection release request from the adjoining layers. The

⁶ The term bypass used here is different from that used by OSI to mean the total removal of a layer or sublayer from a protocol stack. In x-kernel [142], the term bypass is used to prevent the need to execute certain microprotocols in the protocol graph, depending on the type of messages. The term bypass is used here to refer to a way of avoiding a certain set of *functions* in a protocol layer or stack.

Figure 12 Time Sequence Diagram

bypass system identifies incoming data PDUs that keep the entity in the data transfer phase (OPEN state) and processes them separately from the normal processing path.

The ability to separate the normal processing path of data PDUs from the connection establishment and release phase with the data transfer phase, offers the following advantages:

- Processing path of data PDUs can be optimized;
- The number of possible PDU formats in the bypass path is reduced to data transfer PDUs;
- The finite state machine of the protocol is now reduced to only the "OPEN" state, for as long as processing remains in the bypass path. The state of the system does not

Figure 13 Finite State Machine of TP0/TP2

change during the entire data transfer phase and the protocol processing is reduced to ensuring reliable transfer of data across the communications network.

5.2 Bypass Architecture and its Implementation

Figure 14 illustrates the architecture of a bypass system. The standard protocol stack (SPS) is the normal processing path taken by all PDUs during a connection without the bypass implementation. The SPS refers to a single layer or multiple adjoining layers of a layered protocol stack, e.g. the upper layers of the MAP/TOP [133, 134, 182] functional stack. The bypass implementation includes 4 key components namely;

- ☐ *Send Bypass Test;*
- ☐ *Receive Bypass Test;*
- ☐ *Bypass Stack;*
- ☐ *Shared data* for access by the *Send* and *Receive Bypass tests*, the *Bypass stack* and the SPS.

5.2.1 Send Bypass Test

Every outgoing Data Unit (DU) is tested before it enters the SPS. The test involves checking that the outgoing user service primitive is the data transfer request service primitive. Each layer in the SPS is then examined to ensure that they are all in the "OPEN" state, i.e. in the data transfer phase. Next, the SPS is tested to guarantee that there are no PDUs in-transit at any layer. This is important to maintain ordered arrival and delivery of PDUs. If there are any in-transit PDUs encountered in the SPS whilst the outgoing DU is a data transfer request service, the DU is queued until all the in-transit PDUs are serviced.

5.2.2 Receive Bypass Test

Similarly, every incoming DU is tested for bypassable PDUs by the Receive bypass test. This test involves matching the incoming PDU header with an expected header template for PDUs in the data transfer phase for the SPS. The system is then tested for "OPEN" state and "no in-transit PDU" conditions for each layer, similar to the Send Bypass test. This is analogous to Jacobson's "Header Prediction" on receiving TCP packets which requires only a few simple compare operations when packets arrive in the order expected. Incoming DUs that match the header template, but have in-transit PDUs in the SPS, are queued till all in-transit PDUs are serviced.

Figure 14 Bypass Architecture

5.2.3 Bypass Stack

The Bypass stack includes the protocol processing part of the SPS for the data transfer phase. This is a subset of the full SPS. Once PDUs have passed the Send bypass test, protocol processing similar to that of the full SPS is performed by the bypass stack. This ensures that the bypass system fully conforms to the SPS. Next, protocol headers which are exactly the same as would be produced by the SPS, are prepended to the outgoing PDU. Lastly, the service primitive for the SPS provider is attached. Since most of the control information fields of the PDU do not change during the data transfer phase, a header template can be created very quickly and hence speed up generation of protocol headers. The reverse process is true for PDUs that passed the Receive bypass test. Since the Receive bypass test matches the header with a template, the bypass stack knows exactly what protocol operations it needs to perform. Finally the header that matched the template is then deleted and a service primitive attached for the SPS user.

5.2.4 Shared Data

The Send bypass test, Receive bypass test and the bypass stack need to share some state information with the SPS to maintain state consistency. These include the maximum TPDU size negotiated during connection establishment and their window parameters. It is important to ensure that the SPS is consistent with the bypass stack so that control can be passed back and forth between the bypass stack and the SPS without any loss of state information.

5.2.5 Isolating Data Transfer phase from Connection Establishment and Release phase

The main objective of an efficient bypass implementation is to reduce the execution time needed to conduct the send or receive bypass test, and to process the bypass stack efficiently.

The test overhead can be reduced significantly, if the data transfer path can be completely isolated from the connection establishment and release path, i.e. data PDUs

should flow only through the bypass path for the duration of the connection. The "no-in-transit PDU" test must check all layers of the SPS in the given direction, and could be expensive. However, it is not necessary to use it every time. At the beginning of data transfer on a new connection, it is automatically satisfied. As long as no packet fails a bypass test, it continues to be satisfied. Thus it is sufficient to maintain a flag to indicate this. Once a packet fails, and goes to the SPS, then a full "no-in-transit PDU" test must be performed for each packet until the test succeeds, after which control can go back to the flag. The bypass test must also guarantee the ordered delivery of PDUs after passing through the bypass stack.

As the bypass concept conforms fully to the standard SPS, a bypass system could have either Send bypass only, Receive bypass only or both implemented together. Furthermore, it is possible to allow communication between a system that has bypass implementation with another that is not. This is true because the PDUs entering or leaving a bypass system is identical to the standard protocol system. The only difference being the processing path taken by bypassable PDUs and non-bypassable PDUs.

5.3 Multiple-layer bypass

A bigger performance payoff is obtained with a multiple-layer bypass than with a single layer; it is only slightly more complicated than a single-layer bypass and it avoids four expensive aspects of standard multiple-layer implementations:

- ☐ Overhead of encoding and decoding the interface control information (ICI) passed between layers, which describes the service definitions;
- ☐ Executing the full general protocol logic for the layers to decide how to manipulate the data;
- ☐ Queueing of data at layer boundaries.
- ☐ Context switching.

Multiple-layer bypass has even greater leverage where some layers, like the network and application layers, have been further subdivided into sublayers to distinguish their functionality. These sublayers produce a more structured model, but are inefficient for implementation due to the interaction at the sublayer boundaries.

A multiple-layer bypass is defined by unwinding all the layers in a protocol stack, where there are many possible processing paths depending upon the service primitive, PDU type, current state and any pre-selected options. The bypassed path is a concatenation of processing procedures performed by these adjacent layers when they are simultaneously in the data transfer phase. Meanwhile, the separate layers in the SPS path are able to interact and negotiate its quality of services transparently during the connection establishment phase or initiate a connection release. Figure 15 shows the processing path taken by bypassed packets through a multiple layer protocol stack.

Multiple layer bypass offers the possibility of merging the functionality of the presentation, session and transport layers together, similar to the HOPS architecture proposed by Haas [84, 85]. This gives the implementor the flexibility to optimize the processing path of these bypassed packets. Functions from different layers can be grouped together to increase performance. All per-octet operations can be grouped together and that for each access to the data, as many per-octet procedures is operated as possible.

5.3.1 Difficulties of multiple-layer bypass

An ideal situation is to bypass the complete seven layers of OSI during the Data Transfer phase, but this is not always possible. Flow control mechanisms can be implemented in both the Transport and Data Link layers. Implementing the back to back flow control mechanism of both layers will cause inefficiencies as the upper layers flow control is now dependent upon the lower layers flow control state. It may be better to separate the functional stack between the upper 5 layers (Network and above) and the lower 2 layers (Data Link and below), so that bypassing can be divided into two functionally independent stacks. Furthermore, the processing of these stacks are typically

on physically different hardware which makes this division even more attractive. For multilayer bypass to be successful, segmentation/reassembly is not allowed within the bypass path. However, we do not foresee this as a major problem, as research suggests that fragmentation [117] of PDUs should be restricted only to the lower layers and should occur only once in the protocol stack. The Segmentation and Reassembly sublayer of the ATM adaptation layer is a good place for such functions [130].

5.4 Bypass without window flow control

This section describes briefly the four components of an experimental bypass system

Figure 15 Critical path taken by bypassed packets

Figure 16 Sender and Receiver Header Templates

for the OSI session layer and transport layer class 0. The next section describes in more detail the same protocol stack, but with transport layer class 2, which is more complicated as it contains multiplexing and window flow control mechanism (optional).

Send bypass test — As we wish to bypass the session and transport layer together, the test is conducted on the IDU as it enters the session layer. The first test conducted is to determine whether it is a session data request service primitive, SDTreq. The SDU is then tested to ensure that it can fit into the maximum TPDU size. The session and transport layers are finally tested to ensure that they are in the "OPEN" state and there are "no in-transit PDUs".

Receive bypass test — The Receive bypass matches the incoming IDUs with the header template as shown in figure 16. It must be determined that incoming PDUs were not segmented and hence do not need reassembly. This can be done by examining the EOT bit in the arriving packet. Finally it checks for "OPEN" state and "no in-transit PDUs".

Bypass Stack — The header that is generated by the bypass stack as the SDU passes through it is a concatenation of the session and transport headers. This is similar to the expected header template for data PDUs at the receiver (see figure 16). The TPDU-NR field for class 0 is not used here and set to 0. The bypass stack then attaches the service primitive before enqueueing to the respective queues.

Shared data — The shared data contains essentially the status information to test for any in-transit PDU in the session and transport layers and the maximum TPDU size negotiated for the transport layer.

LI	AK 0110	CDT	DST-REF	YR-TU-NR
----	------------	-----	---------	----------

Figure 17 ACK TPDU

5.5 Bypass Algorithm with window flow control

This section describes the bypass elements for the OSI session and TP2 layer SPS. With explicit flow control, the bypass stack has to ensure that the window credit is not exhausted before a TPDU can be sent. In the Receive bypass path, it has to determine whether an acknowledgment (AK) TPDU need to be sent. The AK TPDU (figure 17) contains a "new credit" field and acknowledges that all the packets up to YR_TU_NR - 1 were successfully received. The frequency for generation of AK TPDUs are left to the implementor, but some restrictions, such as no credit reduction, apply. On receiving an AK TPDU, the sender has to update the lower window and upper window edges.

The Send and Receive bypass test is similar to that described in section 5.4 without window flow control, except that the header template for the receive bypass test is now as shown in figure 18. The TPDU-NR is masked out as its value changes with each data (DT) TPDU. Here, the bypassed packets are predetermined to contain DT TPDU that uses the normal format and there are no variable fields. This simplifies the processing required by the bypass stack.

The shared data include those described in section 5.4 and also the window parameters for both the sender and receiver.

Multiplexing is done on the Receiver side by reading the DST-REF field of the incoming packets, and hence they can be channelled to their upper layer protocol entity via their corresponding service access point (SAP). On the Sender side, the bypass stack knows the DST-REF (from the shared data component) of the corresponding SRC-REF

as these parameters were determined in the connection establishment phase. Multiplexing is also handled the same way for ACK packets.

The pseudo 'C' code algorithm for bypassing with window flow control for the send and receive path is presented Appendix A

5.6 Experimental Setup

A multiprocessor testbed for an all-software prototype was constructed [79]. It consists of the application process that generates and consumes packets, a session layer that supports the basic combined subset [98], a transport layer with classes 0 and 2 [104], and a vestigial network layer. Figure 19 shows the structure of this experimental testbed. The hardware was a multiprocessor system consisting of 12.5 MHz MC68020 single board computers with 1 Mbyte of 1-wait-state DRAMs, connected via a VMEbus [8].

5.6.1 Software structure

Figure 20 shows the software procedural flow chart for the protocol testbed. It consists of six procedures and a table driven finite state machine (FSM). Packets arriving at a layer interface are queued in two separate queues depending on whether it is arriving from the layer above (SDU IN queue) or the layer below (PDU IN queue). When the *Main* procedure receives a message, it calls either the *Process_Physical_In (PPI)* procedure or the *Process_Application_Down (PAD)* procedure. If the message arrives from the application layer, the procedure *PAD* is called. Otherwise, if it arrives from the

Figure 18 Header Templates

start the layer. *RT* starts up its FSM and then calls the procedure *Process* to process the message containing the PDU or SDU generated by the FSM. The FSM corresponds to the official semi-formal OSI specification, which are represented by state tables that have a form of a matrix of the type *event X state*. For each possible combination of an event and a state, additional conditions may be specified in the form of predicates, and, for each condition, the actions to be taken and the next state. The FSM uses several procedures to: 1) decode the current state conditions, 2) find the appropriate state transitions and 3) execute the appropriate actions, which include the construction/stripping of the header for the outgoing SDU or PDU. The procedure *Process* extracts the PDU or SDU from the received message and moves it to the appropriate queue. It then starts the next layer (up or down) via a call to *SL*.

For packets travelling up the protocol stack, *Main* calls *PPI* sending it a message that contains a PDU. *PPI* extracts the PDU from the message and enqueues the PDU in the network layer's PDU IN queue. It then calls *SL* to start up the network layer. *SL* then calls *RT* sending it the state of the current layer. *RT* calls the FSM sending it the PDU and the state information. The FSM returns the newly generated PDU via a message to *RT*. *RT* then calls *Process* which extracts the PDU from the message and enqueues it in the transport layer's PDU IN queue. Next, *Process* calls *SL* to start up the transport layer. This loop continues until the PDU has been pushed to the top of the protocol stack.

For packets travelling down the protocol stack, *Main* calls *PAD* sending it a message that contains an SDU. *PAD* extracts the SDU from the message and enqueues the SDU in the session layer's SDU IN queue. It then calls *SL* to start up the session layer. *SL* then calls *RT* sending it the state of the current layer. *RT* calls the FSM sending it the PDU and the state information. The FSM returns the newly generated SDU via a message to *RT*. *RT* then calls *Process* which extracts the SDU from the message and enqueues it in the transport layer's SDU IN queue. Next, *Process* calls *SL* to start up the transport layer. This loop continues until the SDU has been pushed to the bottom of the protocol stack.

Figure 20 Procedural Flow Chart of Protocol Testbed

The "couriers" are tasks used to transport messages and also to prevent deadlocks as the protocol software must be capable of handling traffic in both directions. Interprocess communication and synchronization is provided via unison [139] messages. It uses the send, receive and reply abstraction. Messages are blocked on send until it gets an explicit a reply from the receiver, which is similar to the Ada rendezvous mechanism.

5.7 Results of Bypass Implementation

The experiment was initially carried out with one processor for each end of the connection (2 processor version). Another set of experiments was then conducted using two processors for each end of the connection (4 processor version) to isolate the protocol processing part (session and transport layers) from the user process and the rendezvous

message passing mechanism of the courier tasks. Here, the Protocol task was executed on one processor whilst the user and the two courier tasks were executed on another on each side.

Multiple-layer protocol bypass for both session and transport layers were implemented. Tables 3 and 4 shows the results of bypassing the session and transport class 0 and 2 layers. The results were compared between the non-bypassed system and a bypassed system with both the Send and Receive bypass paths implemented. The average throughput measurement in packets per second is taken for the transfer of messages for a complete connection comprising 1000 packets of 20 octets each, with the maximum TPDU size set at 1024 octets. This result was then used to calculate the rated throughput for a packet length of eight kilo octets in order not to bias against bus saturation on the VMEbus of the target system. These results were further extrapolated for current RISC based processors with a rated performance of 10 MIPS. The figure arrived here is a rough estimate, as we assume that the CISC/RISC instructions ratio for the bypass procedures are the same.

In table 4 (Session and Transport Class 2), the 4 processor version only gives an improvement of 6.6% compared with the 2 processor version if there is no bypass. This is because improved throughput by pipelining the user process and the protocol process depends upon the load balancing of these two processes. The throughput is limited by the slowest stage in the pipeline [215], which is the protocol process in this case. When bypass is implemented on the 4 processor version, there is marked improvement of 73.9% (11.40 Mbps to 19.84 Mbps) compared with the same version without bypass. A 85.6% improvement is achieved when compared with the 2 processor version without bypass. The rated throughput (for 8 kilo octet packets) of a 8 processor version with a pipeline architecture without bypass is measured at 17.8 Mbps [207], compared with 19.84 Mbps for a 4 processor version with bypass. In the 8 processor version, 4 processors were used for each end of the connection with a processor for each of the 3 layers and the

user process. The workload is again not evenly distributed as the session and transport layers perform much more processing than the network layer (vestigial) and the user. Another reason is that bypass attacks interlayer processing whilst a pipelined architecture may increase it due to interprocessor communication overhead. Here, bypass outperforms a simple parallel pipeline architecture for protocol processing. It is quite possible that packet level parallelism would be better, but it shows that bypass gain is in the useful range. The same argument holds for results given in table 3 (Session and Transport Class 0). An analytical model using Petri Nets for this pipelined version of the Session layer and TP2 was presented in [208] by Woodside *et al.*

In the 4 processor version (Session and Transport Class 2), by selecting a particular connection to be non-bypassable, the penalty due to bypass test overheads is about 0.23% for 1000 packets of 20 octets each. Also, bypassing TP2 alone improved performance only by 21%, whilst bypassing the session layer alone improved throughput by a mere 13%.

Session and Transport protocol Class 0 Bypass

Number of Processors	Bypass	Average Packets/sec (20 octets packet)	Rated Throughput for 8 kilo octet packets (Mbps)	Percentage Change (compared with no Bypass on 2 processors)	Extrapolated for a 10 MIPS RISC based processors for 8 kilo octet packets (Mbps)
2	NO	171.0	11.21	0	89.68
2	YES	247.28	16.21	+44.6	129.68
4	NO	206.4	13.53	+20.7	108.24
4	YES	356.8	23.38	+108.6	187.04

Table 3 Bypass of Session/TP0

Session and Transport protocol Class 2 Bypass

Number of Processors	Bypass	Average Packets/sec (20 octets packet)	Rated Throughput for 8 kilo octet packets (Mbps)	Percentage Change (compared with no Bypass on 2 processors)	Extrapolated for 10 MIPS RISC based processors for 8 kilo octet packets (Mbps)
2	NO	163.0	10.69	0	85.52
2	YES	219.1	14.36	+34.3	114.88
4	NO	174.03	11.40	+6.6	91.24
4	YES	302.6	19.84	+85.6	158.68
8	NO	271.44	17.79	+66.5	142.32

Table 4 Bypass of Session/TP2

5.8 Summary

The pure software approach described in this chapter is good for a protocol stack that does not include presentation conversion routines or checksum, as interlayer processing overhead dominates all other protocol processing [201]. However, if per-octet operations are included in the protocols, per-octet operations quickly dominate. Hence, the pure software approach for bypassing may not be very attractive.

In section 4.1, we discussed the implications of Gbps communications and suggested that a 150 MIPS processor is required for a protocol stack that require just 1 machine cycle per octet for X , 200 instructions for B and $P = 1024$ octets (refer to equation 4.1.3). Since the session and presentation layers perform only a simple header construction during the data transfer phase, they do not contribute substantially to B . Unfortunately, even a 150 MIPS processor and its matching memory subsystem (a memory cycle time of 6.69 *nsecs* is needed) is not economically justified with current technology. A different approach is required, and there seems to be two possibilities; VLSI implementation and

parallel processing. These are the ROPE and POBA architectures presented in the next two chapters.

Chapter 6 A Reduced Operation Protocol Engine (ROPE): Design and Implementation of Bypass chip

6.1 Introduction to VHDL concepts

The VHSIC Hardware Description Language (VHDL) [148, 52] is used here to model and synthesize the bypass system. VHDL is an industry standard language (IEEE 1076 — 1987 [1]) used to describe hardware from the abstract to the concrete level. VHDL can be used to represent all levels of abstraction from logic gates to the system level. By utilizing VHDL as a specification tool, it is possible to begin simulation and debugging of complex systems before details regarding the implementation are fully specified. This is especially useful in a complex system such as a communication subsystem. It also offers the potential of an automatic path from the protocol specification to VLSI implementation, in which any modifications to the specification can be easily propagated to the gate level design.

A key aspect of top-down design is its adaptability to emerging and varied technology. With the behavioral description, we can resynthesize for a new technology. It lets the tools build the individual components, and the designer just has to specify the behavior of the system, similar to protocol specification from Formal Descriptive languages like LOTOS and ESTELLE [24]. It may be possible, as part of the proposed future work, to automatically implement the bypass system in VLSI if a formal derivation of the bypass path can be automatically extracted from the formal specification of the full protocol [121]. It can then be passed through an automated translation tool to convert to VHDL code for synthesis.

Figure 21 Software and Hardware processing path of a non-bypass versus a bypass system

6.1.1 Design

Figure 21 compares the way in which the execution of a conventional protocol stack is divided between pure software implementation on the host at the top and special purpose interface hardware at the bottom. With the division for bypass processing, notice that the host/adaptor boundary has moved up during the data transfer phase. During the connection setup and release phases, all PDUs in the bypass stack are processed in host software. During the data transfer phase, protocol execution from the Physical to the Presentation layers can be performed by dedicated hardware. In this study, we would like to focus on the performance of the bypass chip called ROPE, so as not to limit the study to any specific host architecture or network technology. The host processor transmits Application PDUs to the bypass chip, whilst the Network interface adapter (NIA)⁷ receives Datalink PDUs from a high speed network. The Host processor, Network interface adapter and

⁷ Network Interface adapter refers here to the software/hardware that supports lower layer protocols and their interface to the transmission medium. e.g. an FDDI or ATM controller. The term adaptor is used to refer collectively to the software/hardware components that supports the bypass stack and the traditional Network Interface Adapter components. It is assumed that they are both implemented on the same board (tightly integrated).

the bypass chip will synchronize the processing path of PDUs by the send and receive bypass tests conducted at their layer interface. A NULL network layer is assumed here in this study.

6.1.2 Architectural Description

Figure 22 shows the block diagram of the bypass system with the bypass stack implemented in VLSI. The architectural considerations involved in this design can be summarized as follows:

- Movement of data across the host bus interface are minimized using an on-chip DMA for fast block data transfer to/from the host system memory.
- On-chip dual-ported memory is used. The main advantage in providing packet buffering on the bypass chip itself, rather than using the host memory, is that critical constraints on bus access latency and throughput are avoided.
- The control registers of the bypass chip are I/O mapped to the host processor. This enables the host processor to configure the bypass chip directly.

6.1.3 Experimental Setup

Figure 23 shows the flow diagram of our design process. There are three stages: a behavioral model, a structural or RTL model, and a gate level design [5, 127]. These gave us two kinds of feasibility check; that the logic we specified will execute the protocol within the environment we envisage, and that the design is technically feasible, for instance in a reasonable chip area. The SYNOPSYS [186, 187] synthesis tool was used to generate the gate level design (see appendix D). The timing information from the vendor component library was back-annotated to the structural model and resimulated to obtain the throughput results presented in section 6.2. The final step of generating a chip layout for fabrication and fault analysis was not performed as it is beyond the scope of this feasibility study.

A VHDL behavioral model for the system was initially written and tested. It includes the bypass chip, the host processor with a simplified bus/memory architecture and a

Figure 22 Block Diagram of VLSI bypass system

vestigial high-speed network interface adapter which acts simply as an infinite source/sink for data packets. The first design implemented the Basic Combined Subset (BCS) of the session layer and the Transport protocol class 2 (TP2) protocols. Only the logic for the data transfer phase is simulated in this model, as it is during this phase that the bypass chip is active. The host processor submodel generates a simple test sequence that initiates bypass connections and a series of bypassable data packets. Non-bypassable packets are also inserted to simulate for example the arrival of a connection release PDU. The test sequence allow us to verify the following:

- Window flow control logic.

- Generation of the header field including the sequence number.
- Generation of acknowledgment packets on receive.
- Synchronization between the SPS and the bypass stack.

Next, the behavioral model of the bypass chip was manually converted to a structural (RTL) model for synthesis⁸, leaving the other components as behavioral constructs. RTL descriptions are lower level than behavioral descriptions. The distinction is that a behavioral description has no implied architecture in its representation, while an RTL level description has a definite architecture and clocking scheme. They characterize systems in terms of registers, switches (multiplexors), and operations. An initial assignment of operations into clock cycles is also made. These descriptions, like behavioral descriptions, are technology-independent (silicon library independent). In a VHDL RTL level description [125], not all of the functionality of VHDL can be used, because some of the language features do not map into the RTL model. For example a WAIT FOR statement has no meaning in hardware, but is useful in behavioral simulation. Features that can be easily mapped to hardware include IF THEN ELSE statements and signal assignments.

The same test sequence was again generated by the host processor, and its results were compared with the original model to ensure behavioral consistency. The structural model was then passed through the SYNOPSYS synthesis tool for gate level generation with the 0.8 μm BiCMOS macro library from Texas Instruments [192]. As the model was too large for the SYNOPSYS package we were using, it was divided into 3 submodels. The dual-ported SRAM was not synthesized as its gate counts and performance characteristics can be easily extracted from data books

The original model was then extended to incorporate some common transport class 4 (TP4) functionality like checksum and timer activities.

⁸ The author is not aware of any tools available that can automatically convert VHDL behavioral models into RTL models at the time of writing this thesis

Figure 23 Design flow diagram

6.1.4 Behavioral description

The Sequence of Operation in the bypass system is summarized as follows:

- 1) Four high level procedures are made available to the host processor to control the bypass chip, namely: **BYPASS_START**, **BYPASS_DMA**, **BYPASS_SYNC** and **BYPASS_RESTART**. On receiving the first bypassable PDU, the **BYPASS_START** procedure is called. This procedure sets up a bypassable connection by sending information like its initial window flow control parameters and the **DST_REF** field to the bypass chip. These information are stored in a process control block (PCB) for the particular connection in fixed on-chip memory locations and are also accessible by the host (I/O mapped). The maximum number of multiple connections allowed for bypass will be limited by the number of these control blocks allocated in the bypass chip (See figure 24).
- 2) For subsequent bypassable packets, the host processor initiates the **BYPASS_DMA** procedure which checks for free buffer space in the bypass chip and programs the

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

DMA by sending the starting address pointer where the PDU is located, and its total length. The destination address is supplied by the bypass chip. Arbitration for the host processor bus between the host and DMA is provided by the DMAreq and DMAack lines. DMA transfers the PDU into the internal dual-ported SRAM (Static RAM). Buffers are pre-allocated in fixed sizes and are accessed by a simple round robin scheme using a set of buffer pointers.

- 3) The protocol engine polls the status field of a packet to check if there are any data to be processed. If the status is FILLED, protocol processing of the bypass stack can proceed. The dual-ported structure allow protocol processing to proceed concurrently with any DMA transfer to/from the host processor. A precomputed header template is used to construct the header field very quickly.
- 4) Processed in-sequence packets within the transmit window are passed to the network interface adapter⁹.
- 5) Whenever the host processor encounters a switch in the processing path, i.e. from the bypass stack to the SPS, it will issue a BYPASS_SYNC procedure. This procedure will flush the bypass chip of any "in-transit PDU" for that particular connection and return any updated information, for example window control parameters, from the bypass chip to the host in order to maintain state consistency between the two paths. This may occur when it receives a connection release primitive or a session control primitive of the session layer (see section 6.3.1) during the data transfer phase.
- 6) Whenever the host processor wishes to re-enter the bypass path after a switch in the processing path, the host will issue a BYPASS_RESTART procedure which will only pass those information that were updated in the standard protocol stack, like window flow control parameters, back to the bypass chip. Parameters like the DST-REF field which is not changed for the duration of the connection need not be updated. Again this is needed to maintain state consistency.

⁹ It is assumed here that the network interface adapter is an infinite source/sink for data packets.

Host Tag : This tag is set on receipt of a host command, e.g BYPASS_START, BYPASS_DMA, BYPASS_SYNC or BYPASS_RESTART.

STATUS : Indicates the status of the buffer, e.g. EMPTY, FILLING, FILLED or CLOSED.

Figure 24 Organization of internal bypass chip memory

6.1.5 Extensions to include major procedures for Transport Class 4 (Implemented)

This section describes extensions to the original design, which only supports Session BCS and TP2 functionality, to include some common TP4 functionality. The checksum, retransmission on timeout and resequencing procedures for the bypass chip were implemented. Extensions to include the Session layer Basic Activity Subset (BAS) and Basic Synchronized Subset (BSS) in a bypass system is described in section 6.3.1. Procedures for presentation layer conversion within the bypass stack are not implemented, but will be discussed in section 6.3.2.

6.1.5.1 OSI Checksum

The transport protocol class 4 checksum algorithm (See appendix B) was proposed by John Fletcher [75] as a way of providing the same order of protection, in software, as the more computationally expensive CRC algorithm which are typically handled in hardware. Since it is executed in software, it is still somewhat computationally expensive and has a profound impact on the throughput performance of the protocol, especially in bulk data transmission. The XTP checksum functions are straightforward to implement in hardware, but the ISO TP4 checksum function is not, because it is byte-oriented and involves modulo 255 arithmetic. In bypass, checksum of small control packets (non-bypassable) are done in host software, as the overhead incurred by checksum for small packets is small compared to other processing overhead like context switch and buffer management [116].

It is difficult to perform OSI checksum on the fly at the sender end as the two byte checksum field is placed in the variable part of the header. However, with the bypass system, the position of the checksum field is fixed and hence calculation of the checksum field can be simplified further. This is accomplished by precomputing the partial checksum of the header fields.

6.1.5.2 Timers

During the data transfer phase TP4 uses a number of timers, namely the Retransmission timer (T1), Window timer (W) and the Inactivity timer (I). The retransmission timer (T1) is used to maintain a bound on the time it will wait for an acknowledgment before retransmitting the TPDU. It is possible to maintain either one retransmission timer interval per transport connection or one interval per TPDU. The window timer (W) is used to ensure that there is a maximum bound on the interval between window updates providing timely updates of flow control credit information. It is reset each time an AK TPDU is sent. The inactivity timer (I) is to detect when the remote entity has ceased to send anything. This is mandatory when operating over a connectionless network service, since there is no way to determine whether the remote entity is functioning or not. Only the retransmission timer with one interval per connection and the window timer were implemented here.

Timer management in software is an expensive process due to software interrupt handling overhead and update processing of the timer queue [199], but can be easily handled by VLSI implementation. On-chip timers are very efficient and can be executed concurrently with other protocol processes. The only overhead is in starting and stopping the timers. Once it is started, a timer is an autonomous process until an interrupt signal is activated. A separate area of on-chip memory is reserved to store the state information of the timer list.

6.1.5.3 Retransmission and Resequencing

In the receive end, out of sequence PDUs outside the transmit window will be discarded. Otherwise, it is buffered for resequencing. Duplicate TPDUs can be detected easily because the sequence number matches that of previously received TPDUs. On the sender end, if timer T1 expires, the transport entity can either retransmit only the first TPDU or all TPDUs (Go-back-N) waiting for acknowledgment. In this design the Go-back-N retransmission strategy is used. For large window, the on chip buffer may

not be sufficient to hold the unacknowledged data packets for retransmission or to buffer data packets for resequencing. Slower external memory will be needed.

6.2 Results

The timing information obtained from the netlist is back-annotated to the structural model to obtain throughput performance results. The operating parameters and assumptions made in this study are:

- ☐ 1 Kbyte data packets.
- ☐ Window size of 64. Acknowledgment packets sent for every 20 packets received. The window is always open.
- ☐ The host bus/memory subsystem and network interface adapter are assumed to be infinite sinks/source of data packets.
- ☐ One thousand packets are processed for each iteration.
- ☐ 4 Kbyte of internal dual ported SRAM.

Table 5 shows the throughput performance of the bypass chip with Session layer (BCS) and TP2. The result includes the time taken to move the data packet out from the internal memory of the bypass chip to the network interface adapter, but not the data copy operation from the host memory. Hence the throughput result includes one copy operation of the data packet. The total gate count for the bypass chip with Session (BCS), TP2 and 4 Kbyte internal dual ported SRAM is 51,231 equivalent NAND2 gates. With additional TP4 procedures like checksum and timer circuitries, the total gate count increases to 54,545 gates. Texas instrument offers a gate array package with 112,000 usable gates (TGB1150) [192]. This leaves enough chip area for extra on-chip memory or hardwired presentation procedures. With no per-octet operations, i.e. just protocol processing of packet headers (TP2), the bypass stack could effectively achieve a throughput of 2.362 Gbps. This is consistent with results presented by Clark [44] which claims that TCP processing can achieve up to 800 Mbps (without checksum) on current RISC-based

<i>Procedures</i>	<i>Combinational Area (Equivalent NAND2 gates)</i>	<i>Non Combinational Area (Equivalent NAND2 gates)</i>	<i>Total Area (Equivalent NAND2 gates)</i>	<i>Throughput performance with 1 Kbyte packet length (Mbps)</i>
<i>Session (BCS)/ Transport Class 2</i>	6318	2929	9247	2,362.8
<i>Dual Ported SRAM (4 Kbyte)</i>	N/A	N/A	<i>Approximately 41,984</i>	N/A
<i>Session (BCS)/ Transport Class 4 with the addition of the Checksum Procedure with timer circuitries</i>	8334	4227	12561	313.3

Table 5 Throughput Performance and gate count of bypass VLSI chip (ROPE)

processors. With OSI checksum, the throughput performance drops to 313.3 Mbps. This is still more than an order of magnitude higher than an equivalent 19.2 Mbps optimized software hand assembled implementation of the OSI checksum algorithm on the VAX 8800 [170].

6.3 Extensions to support Session BAS and BSS, and the Presentation layer (Not Implemented)

This section discusses extensions to the bypass system to support Session BAS and BSS, and the Presentation layer.

6.3.1 Session BSS and BAS

The session layer [97, 98] is defined by a number of service subsets, comprising of a selection of functional units. The Basic Combined Subset (BCS) includes the kernel, half duplex and full duplex functional units. The Basic Synchronized Subset (BSS) includes the functional units of the BCS together with minor and major synchronization functional units. Communications between Session service (SS) users may be structured in the form of a series of dialogue units. Each dialogue unit is identified by a pair of major synchronization points. These dialogue units allow data exchanges to be totally separated from each other. Using these services, users may identify the start and end points of a dialogue unit, and resynchronize their data exchanges, when necessary. Each major synchronization point has an associated serial number, called synchronization point serial number. Serial numbers are assigned by the SS provider, but their association with synchronization points is made known to both users. This allows SS users to subsequently make a reference to a major synchronization point, and resume communication from that point onwards whenever necessary. This procedure is termed resynchronization and is supported by the S-RESYNCHRONIZE primitive. Minor synchronization points are used by presentation entities to further structure their communications, but they are embedded within a dialogue unit. Minor sync points are more flexible than the major sync points. For example, each minor sync point may or may not be confirmed, and it is also possible to resynchronize to any minor sync point within a dialogue unit. A confirmation confirms all previous minor sync points. Each minor sync point is also identified by a synchronization point serial number for later reference. Synchronization routines are particularly useful when transferring large data files over unreliable networks.

The Basic Activity Subset (BAS) provides the services for activity management between two cooperating Session Service Users. Activity management provides a means to start, end, resume, interrupt or discard an activity. The most important distinction between the activity and synchronization mechanism is that, unlike a dialogue unit, an

Figure 25 Activity management, Dialogue units and Synchronization points

activity may spread over a number of session connections. An activity may be interrupted over a connection and subsequently resumed over a different connection. Each connection is identified by a Session Connection Identifier. It is possible to resume an activity at any of the confirmed minor or major synchronization points. Figure 25 shows the distinction between activity and dialogue management in a session layer operation.

In the BSS, the session services do not actually save session SDUs and do not perform the recovery operations. These activities are the responsibility of the application layer or the application program. The session layer merely decrements the serial number back to the synchronization point and the user must apply it to determine where to begin recovery procedures. Hence these activities are best handled by the standard protocol stack on the host processor. This will also reduce the complexity of the bypass chip thereby maintaining a lean path. However, the penalty paid here is the processing overhead needed to switch between the SPS and bypass stack. Since the number of synchronization points inserted during the data transfer phase is typically small and can

be controlled by the application program, the number of switches in the processing path of the bypass system can be minimized. Similarly, in the BAS, the activity synchronization points can also be controlled and minimized.

6.3.2 Presentation Layer processing

During the data transfer phase, the bypassed presentation layer processing consist only of the Presentation data PPDU (TD PPDU) encoding/decoding functions. Hence the amount of coding required is simplified in the bypass stack. This makes it more amenable to VLSI implementation. However, a potential problem in implementing the presentation conversion routines in hardware is the difficulty in changing its design once the chip is made.

The OSI presentation layer offers a context negotiation capability that enables it to establish a common set of data representation techniques with its peer entity. Each technique, or defined context, is identified by a pair of values, an abstract syntax name and a transfer syntax name. These agreed upon techniques form a defined-context set. The transformation, or encoding, of data values of an abstract syntax into a sequence of octets is achieved through the application of a transfer syntax. Currently, ISO and CCITT only defined one abstract syntax and one transfer syntax: namely ASN.1 [178] and the Basic Encoding rule [193]. However, other sets of abstract and transfer syntax — like encryption and compression algorithms — are likely to be developed.

Each application layer APDU will require a corresponding encoding/decoding function depending upon the defined context set. Hence, it is not feasible to develop a dedicated hardware for each separate APDU. Furthermore, different host system architecture and programming languages used to generate the data structures will have differing encoder/decoder routines. Hence, these presentation procedures are better handled in software (flexible). One possible application of ROPE, which includes hardwired presentation conversion, is in video servers which uses ISO proposed standards such as MPEG [80].

6.4 Summary

It can be concluded from this study that it is feasible to implement the bypass stack (at least for the transport and session layers) in VLSI and that its performance would be at least an order of magnitude higher than pure software methods. The bypass system offloads the critical protocol functions and their associated non-protocol related functions onto a Reduced Operation Protocol Engine (ROPE). The gate count for the bypass chip can easily fit into a commercially available gate array Integrated Circuit. By offloading per-octet type operations from the host, it leaves the host with extra cycles for the application program. The host processor is also relieved of acknowledgment processing which can be expensive as it is often accompanied by an interrupt to the host processor.

In the first design, the bypass chip can support a throughput rate of 2.362 Gbps (Session BCS and TP2) for 1 Kbyte TPDU packets using current $0.8\mu m$ BiCMOS technology. Extensions to TP2 procedures to include TP4 checksum, retransmission on timeout and resequencing procedures decreased the throughput to 313.3 Mbps.

The next chapter proposes an alternative approach which is software configurable and provide a pool of processors each having their own separate local memory where data packets are stored and processed. This architecture allows concurrent memory accesses in each processor, thereby reducing memory access latency in per-octet operations.

Chapter 7 POBA — Parallel Optimistic Bypass Architecture

7.1 Description of POBA

The final design which was considered was a multiprocessor off-board adapter, based on a commercial digital signal processor (the TMS320C40). The architecture is called POBA. It is significantly different from other proposed packet level parallel protocol processing techniques [111, 106] is that only bypassable packets are channeled through the off-board processor.

With a packet size of 1024 octets, a sustained throughput of 1 Gbps translates to a packet being processed or transferred once every $8.192 \mu\text{secs}$. In a single processor version, the processor must be able to handle all protocol processing — including data movements in and out of the processor cache — in that time frame. Consider the case where there is a pool of N processors, each of which is capable of performing protocol processing on each arriving packet independently. Using a simple deterministic round-robin scheduling scheme to schedule arriving packets to the processors, the next packet that is scheduled for the first processor will not arrive till $N * 8.192 \mu\text{secs}$ later. Hence each processor now has $N * 8.192 \mu\text{secs}$ to process a packet of 1024 octets in order to achieve a 1 Gbps rate.

Historical data has shown that VLSI fabrication technology has increased in speed by a factor of about ten each decade [96]. This improvement has been achieved largely by advances in circuit technology. Parallel processing allows us to achieve much higher performance than that possible on a conventional Von-Neumann uni-processor system using the current circuit technology. Powerful low cost general purpose microprocessors

are now available which offer the opportunity for parallel processing [89]. Below is a list of issues which are crucial to parallel processing implementation performance:

- Algorithms — Is the Algorithm Parallelizable?
- Software/Hardware Mapping — How well does the software design map onto the hardware architecture?
- Tight coupling versus Loose Coupling — Shared or local memory systems and Interprocessor Communication mechanism;
- Synchronization — Cooperating processes in a multiprocessor environment must often interact to synchronize and exchange data;
- Granularity and Communication overhead — Communications overhead for small granularity systems must be kept to the minimum, otherwise the performance benefits of parallel processing is lost.

7.1.1 Hardware Architecture

Figure 26 shows the architecture and the operations during a Send. There are N Protocol Processing Nodes (PPNs) with dual-ported SRAM local memory to store a packet for per-octet processing and header construction. A bypassable packet is loaded into the memory through the DMA bus from the host, and is transferred to the Network Interface adapter (NIA) via DMA over a second shared bus linking the PPNs. Shared parameters for a connection must be accessible from the host and the control processor, and are stored in a shared memory SM1. For access from the host SM1 is ported to the host bus; for access from the control processor, it is ported to the control processor local bus. Another shared memory (SM2) links the control processor and the NIA. The rationale for PPNs is exactly the same as in Jain *et al* [111] which only considers the transport layer. Packets are processed by a set of PPNs in a load sharing sequence (such as round-robin) to give packet level parallelism.

The memory and bus architecture is basically driven by the need to support intensive per-octet operations, e.g. presentation procedures, out of local memory. A global shared

memory would be a severe bottleneck [82]. The two buses then separate the two flows which load and unload these local memories to avoid bus bottleneck. The small global shared memories have only once-per-packet access, associated with connection parameters. The control processor provides mutual exclusion between the PPNs. Mutual exclusion with the host is provided implicitly by the bypass test. Access to SM1/SM2 is via the control processor local bus to avoid any further loading of the host bus, which implicitly may have many other functions.

7.1.2 Sequence of Operations

The sequence of operations¹⁰ for Sending packets in a connection is as follows:

- 1) The host processor performs user program and the application layer protocol processing;
- 2) The *send bypass* test checks for bypassable packets at the presentation service interface;
- 3) If the test fails, the packet passes through the Standard Protocol Stack (SPS) in the host, and eventually is transferred to the NIA in the normal manner (non-bypassable);
- 4) If the test succeeds, the host writes the host connection-id and the length of the data unit into SM1. Data Packets in the application address space are then transferred directly to the PPN local memories via DMA1. The destination address — buffer location — is supplied by the control processor via shared memory (SM1). DMA1 is assumed here to have scatter-gather capabilities. On completion, DMA1 interrupts the control processor. The control processor will update its process control block to signal that a new packet has been stored. The buffers are numbered in order, so that its address location can be easily retrieved. In this discussion, we only describe the data flow for a single connection for clarity. Multiple connections are handled in POBA by the control processor which sets up a process control block for each connection. Hence, the state of each connection can be accessed by pointing to the

¹⁰ Here, we only consider bypassable data packets. Connection establishment and handling of session synchronization primitives are discussed in the previous chapter.

relevant process control block. Buffers are allocated to incoming packets from a buffer pool. This allows for better PPN utilization than a strict round robin scheme, to cater for variable packet size and complexity of per-octet operations in various connections. Buffers are released once processing is completed and its data are transferred out to the NIA;

- 5) *Shared data* for the connection are transferred to/from SM1, whenever a switch in processing path takes place. All communications between the host processor and the control processor is via SM1. Communication is not needed between the PPNs and the host as the control processor maintains the current state of all processing in the bypass path. Current window flow control parameters are examples of this shared data required to maintain state consistency between the SPS and the *bypass stack*;
- 6) Each PPN performs all protocol processing for the common case of the bypass stack. Communication between the control processor and the PPNs is via memory on the shared bus;
- 7) The control processor also performs shared functionality like timer management, sequence number generation, acknowledgment processing and retransmission. A flag is used to signal completion of packet processing for each packet.
- 8) Once protocol processing is completed and window credit is available, the control processor will schedule the transfer of packets to the NIA, by passing a pointer to the memory location of the completed packet (PPN local memory) to DMA2;
- 9) DMA2 transfers the completed packet to the NIA FIFO via the shared bus. Ordering is maintained by the control processor.

The receive operation is very similar to the send. The bypass test is now conducted by the NIA. Communications between the NIA and the control processor is also through a common shared memory area (SM2). Non-bypassable packets are channeled to the host via the normal processing host-NIA path. The receive bypass test is also set up to detect ACK packets [196] which are an integral part of data transfer with flow control. The control processor also handle functions such as generation of ACK packets and resequencing. All the functions described thus far are per-packet and per-group-of-packets type operations and hence for bulk data transfer with large packets, the control processor will not be a serious bottleneck (see section 7.2).

7.1.3 Protocol processing at each PPN Node

The following objectives were pursued to attain an efficient implementation in each PPN:

- ☐ The number of machine cycles required for per-octet operation should be minimized;
- ☐ Reduce data movement to the minimum;
- ☐ Exclusive access to a resource should be exercised only when needed;

Multiple layer bypass offers the possibility of merging the functionality of the presentation, session and transport layers together. This gives the implementor the flexibility to optimize the processing path of the bypassed packets. Per-octet operations require access to the data which is stored in the local memory of the PPNs. Each operation requires a read access, followed by the protocol operation itself and possibly a write back to the memory. All per-octet operations can be composed together so that for each access of the data, per-octet processing can be executed together. This is similar to the integrated layer processing concept proposed by Clark and Tennenhouse [43], but in POBA ILP can be done in parallel and for a standard protocol stack. The use of dual-ported memory and DMA allows packets to be transferred to/from the PPNs quickly, while the PPNs can concurrently perform protocol processing.

In the receiver *bypass stack*, processing proceeds *optimistically* by allowing protocol processing to continue before the validation of the PDU is completed. This enables checksum procedures to be conducted together with presentation decoding before the incoming PDU is even validated. In the event that checksum fails or that it is a duplicate, the PDU could be discarded by releasing the buffer.

7.1.4 Advantages of POBA

POBA offers the following advantages over parallel architectures that implement the full protocol logic in each path [111, 106];

- The reduced complexity and the efficient partitioning of protocol functions by bypass, will result in an increase in system efficiency. PPN utilization can be significantly increased due to a reduction in shared memory contention. Various approaches for per-packet level parallel processing [106, 21] have shown that processor blocking due to contention for shared connection state information limits speedup and scalability. An important criteria in the design of a parallel system is to ensure high utilization of processors, whilst reducing communications overhead. POBA offers both possibilities;
- Increased efficiency reduces the number of processors required to support a particular throughput. In our opinion, 10 processors will be the practical limit of the number of PPNs. Otherwise it may not be an economically viable proposition. Server nodes will benefit from such a scheme;
- Multiple-layer bypass allows protocol functions to be merged during the data transfer phase. It makes packet level parallel processing more attractive as packet processing time is increased due to the increased workload for each packet. The increase in granularity will proportionately reduce the communication and scheduling overhead of each packet.

7.2 Characterization of processing requirements and throughput analysis of POBA

In order to match the transmission speed of optic technology moving at a raw data rate of 1 Gbps, all intermediate processes along the path taken by the data unit must be able to process and transfer it at this rate. Otherwise, the throughput will be limited by that of the slowest process and it will become the source of the communications bottleneck. Here, we suggest that the upper layer protocol processing (Transport and above) is the source of bottleneck for high speed communications.

<i>Layer</i>	<i>Procedure</i>	<i>Protocol Processing Node (PPN)</i>	<i>Control Processor</i>	<i>Host</i>	<i>Per-Octet (X)</i>	<i>Per-Packet (B)</i>	<i>Per-Group-Of-Packets Aggregated to Per-Packet for bulk data transfer (B)</i>	<i>Remarks</i>
Presentation	Encoding	X			X			
	Encryption	X			X			
	Compression	X			X			
	Context Alteration			X			X	
Session	Synchronization Management			X			X	
	Token management			X			X	
Transport (Class 4)	Checksum (Optional)	X			X			
	Tuner Management		X			X	X	Depends on Implementation
	ACK processing		X				X	
	TPDU numbering		X					
	Resequencing		X			X		
All 3 layers	Header Construction	X				X		
	Header Decode		X			X		
	Buffer Management		X			X		Minimized (Simple scheme)
	Data Movement	X			X	X		Use of dual-ported memory and DMA. With multiple-layer bypass, data movement is minimized with the PPNs accessing its local memory.

Table 6 Partitioning of protocol functions to host, PPNs and the Control processor

7.2.1 Workload Analysis for POBA

Table 6 is a summary of the procedures that are performed by the PPNs, the control processor and host processor during the data transfer phase. All per-octet operations are handled by the PPNs, whilst per-packet and per-group-of-packets procedures that require synchronization between packets are handled by the control processor.

Assuming the host processor, host bus/memory bandwidth and the NIA can support

Figure 27 Bus Utilization

throughput rate of up to the bus transfer rate of POBA, there are 4 potential sources of bottleneck within POBA listed as follows:

- Shared Bus bandwidth;
- PPN processor speed;
- Control processor speed and;
- Memory cycle time — Up to a maximum of the bus bandwidth.

Figure 27 illustrates the utilization of the shared bus in POBA. It is a potential source of bottleneck for large data packets. In order to achieve high throughput rate, the objective is to reduce the bus idle time in order to maximize its utilization.

The throughput analysis for POBA is different from that of a single processor (see section 4.1) as the per-packet operations B is now divided between the PPNs and the control processor. Let,

C_m = memory cycle time.

B_{ppn} = Average number of machine cycles required for per-packet type operations in each PPN,

B_{cp} = Average number of machine cycles required for per-packet and per-group-of-packets type operations in the control processor.

B_{isr} = Number of machine cycles required to setup and service interrupt routine following a DMA transfer on the shared bus, and

N = Number of PPNs.

and again let,

X = average number of machine cycles required per octet,

P = packet size, and

C = processor cycle time (for both control processor and PPNs)

As in equation 4.1.1, the time taken to process a packet for each PPN, T_{ppn} , is given by,

$$T_{ppn} = C.(X.P + B_{ppn}) \quad (7.2.1.1)$$

For the control processor,

$$T_{cp} = C.B_{cp} \quad (7.2.1.2)$$

The time taken to transfer a packet via DMA in shared bus, T_{dma} , is given by

$$T_{dma} = \frac{4.B_{isr}.C + P.C_m}{4} \quad (7.2.1.3)$$

for a 32 bit (4 byte) bus. The time taken to transfer a packet from host memory to the PPN local memory is omitted for clarity. It can be calculated using the same equation as 7.2.1.3. In the cases studied, it was always less than the time taken to transfer a packet via DMA in shared bus for all P , and so never gave a binding constraint. Here, we assume that the host memory cycle time and the cycle times for the dual-ported SRAM ports are all equal to C_m .

7.2.2 Performance bounds

In place of one bound λ_{max} in section 4.1, we have throughput β with three bounds as follows. Equations 7.2.1.1 to 7.2.1.3, give constraints on β in packets/sec, which can be expressed as,

$$\beta \leq \beta_{ppn} = \frac{N}{C(X.P + B_{ppn})} \quad (7.2.2.4)$$

$$\beta \leq \beta_{cp} = \frac{1}{C.B_{cp}} \quad (7.2.2.5)$$

$$\beta \leq \beta_{dma} = \frac{4}{4.B_{isr}.C + P.C_m} \quad (7.2.2.6)$$

Whichever constraint is the tightest defines the bottleneck of the system. By examining the above equations, we see that PPNs will be a bottleneck if

$$\frac{N}{C(X.P + B_{ppn})} \leq \min \left(\frac{4}{4.B_{isr}.C + P.C_m}, \frac{1}{C.B_{cp}} \right) \quad (7.2.2.7)$$

PPNs are a bottleneck if there are less than a critical number N_{sat} defined by

$$N \leq N_{sat} = \min \left(\frac{4C(X.P + B_{ppn})}{4.B_{isr}.C + P.C_m}, \frac{X.P + B_{ppn}}{B_{cp}} \right) \quad (7.2.2.8)$$

The control processor will be a bottleneck if B_{cp} is sufficiently large or P is sufficiently small, i.e.:

$$\frac{1}{C.B_{cp}} \leq \min \left(\frac{4}{4.B_{isr}.C + P.C_m}, \frac{N}{C(X.P + B_{ppn})} \right) \quad (7.2.2.9)$$

$$B_{cp} \geq \max \left(\frac{4.B_{isr}.C + P.C_m}{4.C}, \frac{X.P + B_{ppn}}{N} \right) \quad (7.2.2.10)$$

These inequalities can be resolved to give a condition on the packet length P , so the control processor is a bottleneck if

$$P \leq P_{sat} = \min \left(\frac{4C(B_{cp} - B_{isr})}{C_m}, \frac{B_{cp} \cdot N - B_{ppn}}{X} \right) \quad (7.2.2.11)$$

Hence for $P < P_{sat}$, throughput is limited by the maximum number of packets that the control processor can handle. Throughout this discussion, "large" packets refer to packet size greater than P_{sat} . One of the primary reasons for building gigabit networks is to enable the exchange of large quantities of data. Thus one would expect to see more large packets (carrying bulk traffic) on the network in the future than we do currently, and so this bottleneck may not be too serious.

The DMA transfer over the shared bus is a bottleneck if C_m or B_{isr} is sufficiently large, i.e.:

$$\frac{4}{4 \cdot B_{isr} \cdot C + P \cdot C_m} \leq \min \left(\frac{N}{C(X \cdot P + B_{ppn})}, \frac{1}{C \cdot B_{cp}} \right) \quad (7.2.2.12)$$

These inequalities can be resolved to give a condition on C_m and B_{isr} , so the DMA transfer over the shared bus is a bottleneck if

$$C_m \geq C_{msat} = \max \left(\frac{4C(X \cdot P + B_{ppn} - N \cdot B_{isr})}{N \cdot P}, \frac{4C(B_{cp} - B_{isr})}{P} \right) \quad (7.2.2.13)$$

or

$$B_{isr} \geq B_{isr(sat)} = \max \left(\frac{4C(X \cdot P + B_{ppn}) - N \cdot P \cdot C_m}{4N \cdot C}, \frac{4C \cdot B_{cp} - P \cdot C_m}{4C} \right)$$

There are several limitations to the above analysis. Processor and bus idle times caused by synchronization mismatch between PPNs, control processor, and the host were not considered. Bus arbitration cycles and host-NIA/control processor communication overhead were also ignored. As β increases, there is also a tendency for the protocol

flow control windows to close due to credit exhaustion: we are assuming the window size is large enough that this will not happen. The packet size on the shared bus for sending packets is actually larger than P , as it contains the header fields. However, this effect is small for large P .

Figure 28 shows the optimistic performance bounds for POBA with 10 processors for the case of $X = 2$, $B_{cp} = 223$, $B_{ppn} = 155$, $B_{sr} = 72$, and $C = C_m = 40ns$, which represents the parameters found for one case (see section 7.3). The PPNs are not constraining.

Figure 29 shows the effect on the throughput bounds by increasing X from 2 to 5, at which point the PPNs are constraining. As X increases, throughput saturates with β proportional to the factor N/X for large packet size if PPNs are constraining. Also,

Figure 28 Performance Bounds

Throughput Bounds for POBA with 10 processors

Figure 29 Effects of varying X and B_{cp} on throughput Bounds

as B_{cp} doubles from 300 to 600 machine cycles per packet, P_{sat} increases. Hence to maximize throughput, packet size can be increased to minimize constraints on the control processor. Another alternative for systems with very large B_{cp} is to use a processor with a smaller C . The effect of doubling B_{ppn} is small for large N and is not shown here.

7.2.3 Scalability

Figure 30 shows the *scalability* of POBA with variations in the PPN workload (XP in machine cycles per packet). Essentially, as the PPN workload increases, we can usefully add more PPNs to maintain the throughput capability. For $P \leq P_{sat}$, scalability is limited by the control processor, and for $C_m \geq C_{msat}$, scalability is limited by the DMA transfer over the shared bus. Substituting $N = N_{sat}$ into equation 7.2.2.4, we get

λ_{ppn} in octets/second defined by

$$\lambda_{ppn} = \frac{4.P}{4.B_{isr}.C + P.C_m} \quad (7.2.3.15)$$

For $P \gg 4.B_{isr}$ and $C \leq C_m$,

$$\lambda_{ppn} \approx \frac{4}{C_m} \quad (7.2.3.16)$$

Similarly, for $P \gg 4.B_{isr}$ and $C \leq C_m$,

$$\lambda_{dma} \approx \frac{4}{C_m} \quad (7.2.3.17)$$

Hence for large P , and $C \leq C_m$, it is possible to scale the system up to the bus/memory bandwidth — which is the goal of POBA.

Figure 30 Scalability of POBA with packet size (P) and PPN workload (XP)

7.3 An implementation using standard off-the-shelf DSP microprocessors

This section describes an experimental study to evaluate the performance impact of POBA using the Texas Instrument TMS320C40 [191] DSP microprocessors for both the PPNs and the control processor.

7.3.1 Proposed Hardware for the adapter

The TMS320C40 processor was chosen to demonstrate the viability of this approach using current standard off-the-shelf microprocessors. It is the first digital signal processor designed specifically for parallel processing. The 50 MHz version runs at a peak rate of 40ns cycle time, delivering up to 275 Mega Operations per second (MOPS). Hardware support for parallel processing includes six on-chip communication ports for synchronous or asynchronous point-to-point interprocessor communication. There are also two separate external bus interface with built-in logic to support mutual exclusion for shared memory accesses between processors. Each PPN uses one of the external buses to access its local dual-ported SRAM. The other external bus is used to configure the shared bus shown in figure 26. Each communication port is controlled by a separate on-chip DMA co-processor and are used for interprocessor communication between the control processor and PPNs. The processor has on-chip 512 bytes of instruction cache and 8K-bytes (divided into two RAM blocks) of single-cycle dual access internal data SRAM. It also has separate internal program, data and DMA coprocessor buses. The internal architecture of these processors supports massive concurrent I/O operations and program execution, and are ideal for per-octet type operations.

7.3.2 Task Architecture

Figure 31 shows the software task architecture for POBA. On sending, the host task checks for bypassable packets. This test can be set up to bypass only certain connections or a particular type of application process. Hence, connections that transmit only a small number of packets or small packet sizes, like remote procedure calls, can be

Figure 31 Software task architecture

made non-bypassable. Whenever a new connection is encountered that is bypassable, the host writes into shared memory (SM1) all the necessary initial shared data information, e.g. window flow control information, host connection ID etc. It then interrupts the control processor (ISR3). In the interrupt service routine (ISR3), the control processor task creates a new process control block (PCB) for the new connection. The maximum number of connections that it can handle is limited only by the memory size. Subsequent bypassable data packets from the host are passed to the local memory of the respective PPNs by reading from SM1 the free buffer pointer. If all the buffers in POBA are used, the host will wait for an interrupt from the control processor that signals a new buffer is available. Otherwise, the host writes the host connection ID and packet length to SM1. It then sets up DMA1 for transfer of the data packet from the host memory to the PPN local memory. Once DMA is completed, it interrupts the control processor (ISR1). On receiving an ISR1, the control processor reads the data from SM1 and updates the respective PCB. It signals the arrival of a new packet which needs to be processed and updates the free buffer pointer by writing the location of the next available free buffer to SM1 (simplified buffer management task).

The main control processor task checks if new packets have arrived. If so, it sends an asynchronous message to the PPN that holds the arrived buffer. This message includes, for example, the sequence number and host connection ID. Asynchronous messages, via the TMS320C40 communication ports, are used so that processors are not blocked waiting for synchronization.

The PPN task checks whether a new message has arrived from the control processor. If so, it decodes the tag and processes the message. For example, the message may signal the arrival of a new packet. It will then update its local PCB. The PPN task also checks periodically for newly arrived packets by testing the status flag. Once the buffer is filled with a new packet, the PPN will start processing. This includes any per-octet operations and header construction. On completion, it sends an asynchronous message back to the

control processor to indicate the completion of a packet. The control processor task also checks for incoming messages from PPNs periodically. It updates its PCB whenever a particular buffer is completed. Since the completion time of the packets may vary, the control processor task ensures that ordered delivery of data packets are maintained. This is done by simply checking the status flag of the next sequence number to send. When the next packet to send is completed and the window is open —not at the upper window edge — the control processor checks whether the NIA is ready to accept a new packet in the same way that the host checks whether the control processor is ready. It then sets up the internal DMA of the PPN to transfer the data out of the buffer to the NIA FIFO. Acknowledgment packets are passed to the control processor through SM2. The NIA processor places the packet in SM2 and interrupts (ISR2) the control processor. The control processor then updates the window information and initiates data transfer for blocked packets caused by a previously closed window.

For TP4 [104, 101], added functionality like retransmission on timeout procedure is handled by the control processor. The go-back-N retransmission scheme is implemented here. A Retransmission timer (T1) which maintains one interval per connection and the window timer (W) (see section 6.1.5.2) is also handled by the control processor.

Notice that interrupts to the host are kept to a minimum as it need not be interrupted for every packet sent. Interrupts on send are typically used to signal the completion of data transfer to the adapter so that buffers can be released, or to signal the availability of the adapter to avoid polling from the host whenever it finds the adapter busy. Since the DMA is slaved to the control processor, the host can set up a list of packets to be passed to POBA and only be interrupted when all the packets in the list is transferred. On the other hand, the control processor is interrupted for each packet DMA'ed. The overhead for handling interrupt service routines in the TMS320C40 are minimized by performing only minimum context saving as only those registers that are used by the interrupt process are saved.

The movement of received packets is essentially the same except in the reverse order. The processing on the other hand is quite different. For example, the control processor also generate acknowledgment packets thus relieving the host from performing this task. For TP4, the receiver also includes functionality for detecting out of order packets (resequencing) and duplicates.

7.3.3 Simulation technique

The above system was simulated using the PARASOL [141, 140] simulator. PARASOL is a "medium-to-fine-grained" simulator/emulator for distributed or parallel computer systems. It can serve both as a performance prediction tool and a prototyping tool for testing parallel or distributed algorithms. The execution times in the PARASOL simulator came from instruction counts from the TMS320C40 software simulator [190], on code compiled by the Texas Instrument C compiler with full optimization turned on. In particular, the per-octet processing and some time critical procedures like interrupt service routines were implemented in assembler to optimize its instruction count. The TMS320C40 offers parallel assembler instructions which can execute two independent instructions in one clock cycle. These instructions were heavily used. Operation times were determined by programming the complete bypass and control logic. The simulator was then used to study the throughput performance of POBA with variations in packet size (P), average number of machine cycles required per octet (X) and the number of PPNs (N). Multiple connection logic was also programmed. Due to the complex interactions between multiple concurrent tasks, the Timeline tool [204] was used for debugging (see appendix E).

The following parameters for the TMS320C40 processor were considered in this analysis [191]:

- 50MHz processor – 40 ns cycle-time (C)
- Memory cycle time – 40 ns cycle-time (C_m)
- Global bus speed — 800 Mbps, 32-bit bus.

- Local bus speed — 800 Mbps, 32-bit bus.
- Link Speed — 160 Mbps, 8-bit parallel port.

Parameter X is an important factor in determining the overall throughput of the system. With $C = C_m$ (no wait states), the value for $X = 3$ for the TP4 checksum (ignoring setup instructions) is achieved by using special instructions like no-overhead repeat loops and single cycle load byte instructions¹¹ provided by the TMS320C40 processor (see listing 1). A similar implementation on a RISC processor takes about $X = 9$ machine cycles as a repeat loop takes 4 machine cycles to flush the pipeline. Also, shifting and masking operations are needed to operate on bytes. This is a factor of 3 worse than the DSP architecture. The TP4 checksum here is more compute intensive than the TCP checksum as it involves the computation of two independent 8-bit checksum fields (see appendix B). The TCP checksum on the other hand is a 16-bit 1's complement addition and is considerably faster. Per-octet operations for presentation conversion was also considered. For example, it takes an average of 13 instructions ($C = C_m$) to encode a random array of 32-bit integers from the ASN.1 [102, 86] format using the basic encoding rule [103], making $X = 3.25$. When the checksum routine is integrated with the ASN.1 presentation conversion routine for the example described above, a value of $X = 4.75$ is achieved. The increase in performance is attributed to the reduction in the number of memory accesses required to load the processor registers, and by exploiting parallel assembler instructions. The benefits of Integrated Layer Processing (ILP) will increase as C_m is larger than C (i.e. more wait states are needed for memory access).

Other presentation procedures that can be efficiently processed by a DSP chip includes bit reversal, byte ordering and EBCDIC to ASCII conversion. While hardware codecs can be used for very efficient coding and compression algorithms, open heterogeneous communication implies that there will not be a single coding technique that will be suited for all data types and all application scenarios. Hence presentation conversion by software

¹¹ It took 7 cycles per octet when compiled directly from the C source code.

seems attractive. With the introduction of lightweight presentation protocols [92] it is possible to further reduce the value of X .

```

; This program shows the inner loop of TP4 checksum excluding
; the setup instructions. Calculate C0 and C1 only
; The TMS320C40 has an instruction that allows zero overhead looping

        RPTED    eloop        ; Delayed Repeat. Zero overhead
        LSH3     -2,R2,R9     ; Setup instructions. Ignore here
        SUBI     1,R9         ; the RPTED instruction needs 3 instructions
        LDI      R9,RC        ; after it

loop:    LBU3     *AR2,AR0     ; BYTE 3
        ADDI     AR0,R0       ; C0
        ADDI     R0,R1        ; C1
        LBU2     *AR2,AR0     ; BYTE 2
        ADDI     AR0,R0       ; C0
        ADDI     R0,R1        ; C1
        LBU1     *AR2,AR0     ; BYTE 1
        ADDI     AR0,R0       ; C0
        ADDI     R0,R1        ; C1
        LBU0     *AR2++,AR0   ; BYTE 0 Post Increment to next word
        ADDI     AR0,R0       ; C0
eloop:   ADDI     R0,R1        ; C1

```

Listing 1

In order to achieve concurrent data processing and data movement, data units are DMA'ed into the on-chip data cache RAM blocks of the TMS320C40 processor initially before processing. Whilst processing on the first RAM block is proceeding, the second RAM block is filled from the random access port of the dual ported PPN external memory (double buffered). Since data are cached internally for per-octet processing, this system scales well even if the current trend of processor speed outpacing memory speed continues, provided the number of clock cycles needed to access a word from external memory is less than $W \cdot X$, where W is the bus width in octets.

7.4 Simulation results

A simplex connection is setup between two identical communicating POBA systems. Five thousand packets were sent for each iteration. The window size was set at 64 and for every 20 packets received, an acknowledgment packet is generated. A propagation delay of $1\ \mu\text{sec}$ was simulated together with a vestigial NIA that supports 2 Gbps transmission rate.

Figures 32, 33 and 34 show throughput versus the number of PPNs (N) and X for 1 Kbyte packets. As the number of PPNs increases (figure 33), throughput increases until N_{sat} . It can be seen that for large X , (in this case $X > 4$), throughput can be increased further by increasing N (exploiting scalability).

Even with more complicated per-octet procedures that make $X = 10$, the system with 10 processors can support up to 200 Mbps.

Figure 32 3D plot of Throughput versus X and N (1 Kbyte)

Figure 33 Throughput versus N (1 Kbyte)

Figure 34 Throughput versus X (1 Kbyte)

Figures 35 and 36 show the average PPN utilization versus the number of PPNs (N) and X for 1 Kbyte packets, which we regard as a "moderate" size. PPN utilization saturates quickly for $X > 4$. The bypass approach has achieved high PPN utilization because synchronization and contention for shared connection information have been minimized, thereby increasing system efficiency. For small values of XP , utilization decreases mainly due to the higher probability of synchronization mismatch as β increases (equation 7.2.2.4). Also, the probability of the windows closing is higher as β increases. As XP increases, PPN utilization reaches saturation. Hence throughput can be improved by increasing the number of PPNs.

Figure 35 Average PPN utilization versus N (1 Kbyte)

Figure 36 Average PPN utilization versus X (1 Kbyte)

Figures 37 and 38 show the control processor utilization versus the number of PPNs (N) and X for 1 Kbyte packets. Comparing figures 35 and 37, it is noted that as the number of PPNs increase beyond $N = 3$, for $X = 1$, the PPN utilization drops whilst the control processor utilization remains constant. This is due to the DMA on shared bus saturation (equation 7.2.2.6). Once the DMA on shared bus saturates, any increase in N will reduce the average PPN utilization without any further increase in throughput as shown in figure 30 ($N = N_{sat}$). As X increases, its effect on throughput depends on PPN saturation. If PPNs are unsaturated ($X < 4$ in figure 35), throughput is unaffected (below the knee of curve shown in figure 34). Once the PPNs are saturated, at the knee, throughput and control processor utilization drop off together as $1/X$.

Figure 37 Control Processor Utilization versus N (1 Kbyte)

Figure 38 Control Processor Utilization versus X (1 Kbyte)

Figures 39, 40, 41 and 42 show throughput versus message length for $N = 2, 4, 8$ and 10 processors (PPNs). From the results, it is demonstrated that it is possible to achieve close to 800 Mbps, which is the host bus/memory bandwidth ($C_m = 40ns$), for $X = 2$ with 8 PPNs. From equation 7.2.2.4, it is shown that throughput is limited by the factor N/X for large P ; For $N = 8$ and $X = 2$, this gives a throughput of $4/C$, which is 800 Mbps in this case. Thus a system based on DSP technology can approach its scalability limit in practice.

7.5 Simulation versus analytical results

Figures 43, 44 and 45 show some simulated results versus the theoretical bound analysis derived in section 7.2.1. Throughput is bounded by the expression in equations 7.2.2.4 — 7.2.2.6 which gives the expected throughput values when no synchronization mismatch occurs between PPNs and the control processor (ideal case). The measured values of $B_{cp} = 223$, $B_{ppn} = 155$ and $B_{isr} = 72$ are used to compare the results.

Figure 39 POBA with 2 Processors

Figure 40 POBA with 4 Processors

Figure 41 POBA with 8 Processors

Figure 42 POBA with 10 Processors

It can be seen from figures 43 and 44 that for large X , the analytical result fits very closely to the simulated results. This can be attributed to the fact that with large XP , the time spent on the PPNs dominate all other processing. In this example, P is relatively large (1 Kbyte). In figure 45, it can be seen that for small values of P , the difference between the predicted and simulated results is bigger especially for small values of X . This is largely due to either control processor saturation or a higher tendency for synchronization mismatch and processor blocking as the rate of flow of packets is increased. The effects of window credit exhaustion, especially for small packet size, will also reduce throughput performance.

Figure 43 Throughput versus X — Simulation results compared with analysis

Figure 44 Throughput versus N — Simulation results compared with analysis

Figure 45 Throughput versus message length
— Simulation results compared with analysis

7.6 Bypass Issues

During the data transfer phase, there is a possibility of control packets arriving in between data packets, for example, session layer synchronization and activity management primitives (see section 6.3.1). These non-bypassable packets flow through the SPS. In order to maintain ordered delivery of packets, all data PDUs in transit, for the particular connection must be flushed and the current state updated before a switch in processing path is allowed. Figure 46 shows the degradation in throughput performance versus the average number of packets per switch in processing path. Since the number of such primitives can be controlled by the application process, the degradation in throughput can be minimized. This effect can be reduced further when multiple connections are active simultaneously since the PPNs are shared amongst all connections and hence its average utilization can be smoothen out. For example, with $X = 4$, $N = 4$ and $P = 1024$,

Figure 46 Effect of switch in processing paths, causing the bypass path to be flushed.

total throughput increases from 79.9 Mbps — for a single connection with 2 packets per switch in processing path — to 155.7 Mbps for two connections, where both the connections also have 2 packets per switch.

More simulation results for POBA are presented in Appendix F. They show throughput performance for different workload patterns and multiple connections. Performance results for 4 Kbyte packets are also shown.

7.7 Predicted performance

It is possible to extrapolate throughput performance fairly accurately, to incorporate for example faster processors, especially for large XP . Figure 47 shows the throughput bounds for POBA with $C_m = 20\text{ ns}$, $B_{cp} = 300$, $B_{isr} = 72$, $B_{ppn} = 155$ and $X = 12$. With $C = 20\text{ ns}$ and $N = 10$, a throughput of approximately 300 Mbps can be obtained

Figure 47 Extrapolation of throughput bounds for improved technology

with a large enough packet size P . With C_m remaining constant at 20 ns, a throughput of about 620 Mbps can be obtained if processors with $C = 10$ ns are available (100Mz clocks). If N is now increased to 20 processors, a throughput of 1.3 Gbps is expected. These predicted throughput are possible even with a large value of $X = 12$, which means a significant amount of per-octet processing can be handled. Depending upon the protocols used and the functionality supported, the value of B_{cp} and B_{ppn} will vary. The effect of any variation in B_{ppn} is amortized proportionately by the number of PPNs. On the other hand a large increase in B_{cp} may make the control processor constraining. Hence a faster control processor may be used under such circumstances.

7.8 Summary

Current off-the-shelf DSP microprocessors are used to evaluate the architecture experimentally with very realistic simulation. It is shown that the system is scalable

up to the host bus/memory bandwidth. The POBA architecture is designed to maximize bypass efficiency. The bypass concept resulted in the architecture being less complex, with a significant reduction in processor contention for shared connection information. Hence its efficiency is significantly higher compared to architectures that implement the full protocol logic [82, 21]. It also minimizes the amount of host/adaptor interaction, thus requiring only a simple command protocol. POBA support multiple connections efficiently with its fair load sharing scheme. Multiple-layer bypass provides extra benefit when combined with packet level parallel processing as it allows protocol functions from different layers to be merged during the data transfer phase.

It is shown that the number of machine cycles needed for per-octet operations can be significantly reduced by the appropriate choice of processors. The DSP chip was chosen as they are designed for fast manipulation of data which are common in signal processing applications and hence a good candidate for per-octet type operations.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis has explored end-system protocol implementation for bulk data transfer at Gbps rates in a standard deeply-layered protocol stack. It has identified and characterized the major software performance bottlenecks associated with protocol processing on an end-system and has applied Software Performance Engineering principles to suggest performance improvements. This led to three techniques — a pure software solution, a hardware solution (ROPE) and a hardware/software solution (POBA) — which were evaluated. For a chosen bulk data stream, the software solution approximately doubled throughput performance. The VLSI approach, ROPE, provides the highest performance for a single processor whereas the POBA architecture is scalable to the bus/memory bandwidth of the attached processors.

Experimental, simulation and analytical studies were conducted to explore and verify our thesis. Each method was chosen to best suit the study. For example, the experimental study was used to verify the bypass concept and the bypass protocol logic. Realistic simulation were used to study throughput performance of both the ROPE and POBA architectures. An analytical model for POBA was constructed to validate the simulation results. It is then used to predict throughput performance of the POBA architecture for any projected improvements in hardware technology.

The performance bottlenecks identified and addressed in this thesis can be summarized as follows:

- Per-octet type operations are identified as the major performance bottleneck for bulk data transfer, and are important even for simple encoding procedures.
- Non-protocol related functions like context switching, interrupt handling and buffer management dominate the per-packet and per-group-of-packet processing. However, their effect can be reduced by increasing the packet size.
- Layered protocol architecture often imposes ordering constraints that make it difficult to optimize the processing path of a multiple layer stack.
- As network bandwidth approaches the memory bandwidth, data movement across the host bus architecture should be kept to the minimum.

In order to achieve a high effective application-to-application throughput performance on the end-systems, it is important to address all the layers collectively. Otherwise, solving one layer in isolation may lead to potential bottlenecks in another. So far, research on upper layer performance issues have concentrated mostly on each layer in isolation. This work is motivated by the underlying philosophy of keeping the critical processing path *simple* even for “heavyweight” standard protocol stacks, as simplicity benefits performance, i.e. making standard protocols “lightweight”.

The software optimization technique for multiple-layer bypass improves performance by mainly attacking interlayer processing overhead. However, per-octet processing time remains a problem in this technique; per-octet functions are best offloaded to an adapter equipped with special purpose processors. However, there are problems associated with offboard processing which must be solved. In POBA, the number of interactions between the host processor and the adapter is reduced, the usage of the host bus/memory resource is minimized and the adapter design is simple.

The hardware/software strategy of POBA is attractive because it offloads not only the dominant workload functions like per-octet processing, but also dominant non-protocol

related functions like context switching, buffer copying and interrupt processing which are poorly handled by current host machines. It uses commercially available hardware and a relatively small amount of code.

OSI protocols were used to demonstrate the effectiveness of this research, due to their rich set of functionality and heavy demands. However, the techniques proposed in this thesis can be generalized to other protocol stacks as its main focus is to support efficient offloading of common protocol functions to an adapter, and to minimize the processing overhead of non-protocol related functions which are present in any protocol implementation.

8.2 Advantages and What it solves

This section summarizes the advantages of the proposed techniques. The SPE principle used to tackle the problem is given in brackets. They are:

- It can make standard protocols nearly as fast as lightweight protocols (Section 3.2), at least for bulk data transfer (*Centering principle*);
- It attacks inter-layer processing overhead (Section 3.3), especially when it is extended to multiple-layer protocol stacks (*Locality principle*). Integrated Layer Processing can be achieved for a standard protocol stack;
- Bypass addresses the key problems associated with off-board processing by selecting a particular set of functions which are complete in themselves for offloading, and requiring only a simple command protocol (*Shared Resource principle*).
- Since bypass identifies a simple subset of protocol processing, it is more amenable to VLSI implementation than an entire protocol stack (*Parallel Processing principle*);
- Parallel Integrated Layer processing is now possible on a standard stack. It applies all the SPE principles to achieve the goal of this thesis. It is the primary contribution of this thesis as it provides a scalable architecture to meet future demands. Since

only the data transfer phase code need to be implemented in POBA, program code space is reduced and its development effort can be simplified;

- Acknowledgment processing is also offloaded (*Processing versus frequency principle*);
- Since per-octet operations can also be offloaded, it is possible to transfer data units directly between the application address space and the adapter via DMA, without any host intervention (*Processing versus frequency principle*). It also reduces the load on the host bus by only crossing it once. Specialized applications like distributed visualization programs can also benefit as it allows application PDUs to be exchanged directly between specialized equipment like video buffers without the need for the host processor to touch the data portion;
- A particular application, for example, bulk data transfer, can be pre-selected for bypassing (*Fixing principle*). Bypass can be initiated by a variety of mechanisms such as an action from the application program or from the quality of service (QOS) parameters [210].
- Since most of the offloaded functions are those that are poorly handled by RISC based processors, like presentation conversion and checksum (Section 3.7), it is possible to choose processors that are better suited for these operations, for example, a DSP processor (*Fixing principle*). One of the reasons cited against offboard processing [146] in workstations is that the offboard processor (assuming a RISC processor is used) is at most the same speed as the host RISC processor. Hence, they argue that it is better to leave all upper layer protocol processing to the host in order to take advantage of any host performance improvements. On the other hand, if a special purpose embedded processor like the DSP is used for these dedicated operations, the potential gain in performance is higher.
- Tradeoff between performance, flexibility and cost is balanced. Flexibility is maintained by processing the complex (and infrequent) task in software and it achieves

performance with hardware support for fast execution of the common functions. The three techniques proposed offer various levels of performance improvements. The pure software approach offers marginal improvement, but at no extra hardware cost. It is also the most flexible. ROPE is more appropriate for workstations as we can place it in the motherboard [136] and integrate it with the LAN hardware. It is the least flexible and is not scalable. The POBA architecture seems ideal for an adapter for a server machine because in this environment the increase in performance and the saving in host bus cycles typically outweigh the cost considerations. It is also flexible (software configurable) and scalable.

- An existing implementation of the OSI stack can be adapted for bypassing without a significant modification of the original software.
- By offloading a large proportion of the protocol processing load from the host, it leaves enough host processing cycles to perform useful work by the application programs.

Speed up of high speed protocol processing has been tried by others. Chapter 4 systematized these efforts according to SPE principles, and then used the same principles to extend the previously defined bypass approach. This structured approach to solving software performance problems led to the elimination of many potential problems. An example that is demonstrated in this work is the importance of applying both the parallel processing and shared resource principles together. Otherwise, the advantages of parallel processing may be offset by synchronization and communication overheads.

8.3 Future Work

SPE principles used here could also be used to improve protocol processing for other types of application like transaction-oriented processing and multimedia communications which may require different optimization techniques. For example in transaction-oriented processing, the data transfer phase is no longer the dominant workload function and hence

the bypass approach is not attractive. In multimedia communications, synchronization criteria for different video, voice and data streams must be met. This may impose additional overhead due to the constant switching between the standard protocol stack and the bypass stack which may affect the quality of service (QOS) delivered by the bypass system. One way to avoid this phenomenon is to include the synchronization mechanism within the bypass path, but that would increase the complexity of the bypass path. Tradeoffs between complexity and performance within the bypass path can be investigated further.

Some possible future directions for this work includes:

- New applications for bypass, for example, Distributed Multimedia Systems (DMS)
- Application of Performance Engineering Principles for fast connection setup in connection-oriented protocols.

These will be discussed briefly.

8.3.1 New Applications for Bypass

The bypass implementation technique gives significant throughput improvements for applications where the data manipulation and data movement operations far outweigh connection establishment, tear down and other control operations. Bypass implementations focus on the data transfer phase rather than the connection management phase. Hence, the benefits of bypass are significant mainly in bulk data transfer and other applications wherein the data transfer phase is longer than the connection management phase. Traditionally this implies mostly file transfer applications. However, advances in fiber optic communication and B-ISDN have stimulated research in Distributed Multimedia Systems (DMS) and many new applications ranging from office automation through medical information systems to video conferencing have become increasingly feasible. These applications involve long-lived connections and bulk transfers of video and voice data. Thus application of bypass techniques to these emerging applications seems both interesting and promising.

Multimedia applications typically involve different types of data objects, ranging from simple numeric and textual data to coded audio, graphic images (static and animated) as well as coded video. These various data objects have different traffic characteristics and delay, throughput and reliability requirements and still have to be handled in an integrated and coordinated manner for presentation to the users. Moreover, these data may originate from physically distributed sources and may be stored or generated in real-time [129].

We believe that recent developments in the architecture and implementation of protocol processing for high-speed networks will have a significant role in DMS. Specifically, our interest is in viewing these techniques and particularly the bypass technique in the light of the two major issues of synchronization and dynamic quality of service management for DMS.

Synchronization Issues

Since most multimedia applications involve interactions between the various media — discrete media such as text or numeric data and continuous media such as voice or video — synchronization of media streams for presentation to the user is an important requirement. Synchronization could occur at various levels of granularity. In tight synchronization, streams of data are usually generated and presented together. The synchronization of audio and video in video telephony called lip synchronization is an example. On the other hand, a need for loose synchronization arises between video and audio when the audio involves only background music. A still coarser synchronization occurs between user interactions and the multimedia presentation. User interactions can be in the form of queries and various editing operations. Typically, the responsibility for tight synchronization lies with the communication sub-system of the DMS whereas loose synchronization is handled by an orchestration manager [23] or a system scheduler, and synchronization with the user interactions is handled at the application or window manager level [128].

In the OSI context, Ramanathan and Rangan [155] suggest that the session layer

is ideal for locating tight synchronization. They propose an extension to the OSI session layer to include a synchronization sublayer. In addition, because fine-grained synchronization presumes media continuity (bounded delays and jitters), they also suggest a sublayer of session layer beneath the synchronization sublayer for ensuring media continuity. This sublayer makes direct use of traditional OSI session layer concepts of activities and dialog units. Since multilayer bypass gains significant performance improvements whenever multiple layers/sublayers exist, application of bypass need to be looked into in such situations.

Loss of synchronization can be caused by various media-dependent delays and jitters in sampling, packetizing, transmission, depacketizing and presentation. If these delays are all bounded, strict synchronization can be achieved by adequate buffering [128]. Transmission delays could be bounded by using connection-oriented, resource reservation protocols which have been proposed in [74, 197, 48].

Quality of Service Issues

In a distributed multimedia system there exists an end-to-end delay between the source and the destination, which may vary due to traffic at the intermediate switches (routers). Variations of delay over an interval are called skews while the instantaneous variations are called jitters. Delays, jitters and skews, besides throughput and reliability, constitute the Quality of Service (QOS) parameters of the communication subsystem. The major contributor to these phenomena is the communication sub-net. Clark *et al* [48] have argued that there are really three classes of applications - rigid applications that need guarantees from the network service provider regarding the quality of service provided, flexible applications that adapt to the existing network condition to raise or lower its quality of service requirements dynamically and those applications that simply are satisfied with a best effort service from the network provider. The first category is called guaranteed service, the second predicted service and the last category best effort service. Predicted services, require dynamic QOS support. For the network

provider to provide any of these categories of quality of service, the applications must indicate its traffic behavior at the time of connection establishment. Based on this information and the current aggregate traffic conditions, the network offers a certain QOS. Thus, admission control is now an important function of connection establishment phase. The complexity of connection-establishment will be significantly influenced by the complexity of admission control. The need for comprehensive admission control decisions at connection establishment time argues in favor of the bypass approach of separating the data transfer phase and handling it in a simplified manner.

The important issue of tight synchronization needs to be addressed with regards to bypass implementations. Specifically, the effect of packet-level parallel processing on the jitter and skews caused in the protocol processing at the end system need be investigated. The effect of dynamic QOS control on the efficiency of bypass also needs further investigation.

Memory to Network connections

Developments in protocol processing for high-speed networks have shown that the real bottleneck is in data manipulation and data movement operations and that the workstation CPU-Memory bus bandwidth is the bottleneck. Hence, it is desirable to provide a direct path between the sources (sinks) of audio/video data and the network interface, avoiding the involvement of the workstation bus in this high-volume, real-time data movement. Off-board processing seems attractive because most of the real-time data is not reusable and hence caching is not beneficial. Also since coding and compression are important requirements for audio and video data, involvement of the host-CPU in such processing intensive operations will drastically affect the CPU availability for application computations.

8.3.2 Applying performance principles for fast connection setup in connection-oriented protocols

An optimistic approach could improve the overall performance of request-response type of transactions. Current connection-oriented protocols provide poor response for transaction-oriented communication [202]. They require explicit connection setup and release, adding significant overhead to simple request-response interactions. Using the Optimistic principle (see section 4.5), we hope to improve the performance of these protocols which will make them as fast as specialized transaction oriented lightweight protocols that were designed for fast connection setup, for example, delta-t and VMTP.

A brief exploration of this topic to show that this goal is feasible, follows. Figure 48 shows a diagram of an optimistic connection setup sequence, where

T_{spcp} — Sender processing time of connection packet

T_{pd} — Propagation delay

T_{rpcp} — Receiver processing time of connection packet

T_{rpap} — Receiver processing time of accept connection packet

T_{spdp} — Sender processing time of data packet

T_{rpdp} — Receiver processing time of data packet

T_{rpackp} — Receiver processing time of acknowledgment packet

T_{sprp} — Sender processing time of Release packet

During the connection establishment phase, the sender entity processes the connection request packet, which is delivered to its peer entity at the receiver. It then waits for a reply from the receiver before it can proceed to the data transfer phase. This exchange of packets between the sender and receiver, which requires complex negotiation for an acceptable quality of service together with the propagation delays encountered by these packets may impose a heavy penalty on its overall performance, especially when there is only one data packet sent during the whole connection. The total propagation delay will become significant for two communicating systems that are far apart. Also, in systems that include more than one connection-oriented protocol layer (OSI), the number

Figure 48 Optimistic Connection Setup

of packets exchanged will be $2 \cdot L$ [218], where L is the number of connection-oriented protocol layers in the stack.

The time spent waiting for a confirmation packet can be used to perform some useful work. The method here is to predict that the negotiation is successful and according to the

requested service. Hence part of the data transfer phase can proceed. The presentation layer can contain very time consuming presentation conversion routines that do not need any extra information from the receiver to decode. The requested presentation context can be used immediately for this conversion. On receiving the confirmation packet, any necessary extra processing needed to complete the data packet like filling the header fields of the destination reference are performed. This extra processing time is denoted by $T_{spec,p}$.

List of Abbreviations

<i>ACK</i>	Acknowledgment
<i>APDU</i>	Application Protocol Data Unit
<i>ARQ</i>	Automatic Repeat Request
<i>ASN.1</i>	Abstract Syntax Notation One
<i>ASN.1-BER</i>	Abstract Syntax Notation One Basic Encoding Rules
<i>ATM</i>	Asynchronous Transfer Mode
<i>BAS</i>	Basic Activity Subset
<i>BCS</i>	Basic Combined Subset
<i>BSS</i>	Basic Synchronized Subset
<i>CI</i>	Communication Interface layer (HOPS)
<i>CISC</i>	Complex Instruction Set Computer
<i>CPU</i>	Central Processing Unit
<i>CRC</i>	Cyclic Redundancy Check
<i>DMA</i>	Direct Memory Access
<i>DMS</i>	Distributed Multimedia System
<i>DNA</i>	Digital Network Architecture
<i>DRAM</i>	Dynamic Random Access Memory
<i>DSP</i>	Digital Signal Processing
<i>DST-REF</i>	Destination Reference (field)
<i>DT</i>	Data
<i>DU</i>	Data Units
<i>EOT</i>	End of Transmission bit
<i>FDDI</i>	Fiber Distributed Data Interface
<i>FIFO</i>	First In First Out
<i>FSM</i>	Finite State Machine
<i>FTAM</i>	File Transfer, Access and Management Protocol
<i>Gbps</i>	Gigabit per second
<i>GSPN</i>	Great Stochastic Petri Net simulation package

IBM	International Business Machine
ICI	Interface Control Information
IDU	Information data unit
ILP	Integrated Layer Processing
ISO	International Standards Organization
ISODE	ISO Development Environment
LAN	Local Area Network
LLC	Logical Link Control
LWP	Light weight Processes
MAC	Medium Access Control
Mbps	Megabits per second
Mbyte	Megabyte
MIPS	Million Instructions Per Second
MPEG	Motion Photographic Experts Group
NAC	Network Access Control layer (HOPS)
NACK	Negative Acknowledgment
NETBLT	NETwork BLock Transfer protocol
NIA	Network Interface Adapter
NRSTind	Network Reset indication primitive
OSI	Open System Interconnection
OSIRM	Open System Interconnection Reference Model
PAR	Positive Acknowledgments with Retransmission
PCI	Protocol Control Information
PDU	Protocol Data Unit
PPDU	Presentation Protocol Data Unit
POBA	Parallel Optimistic Bypass Architecture
PPN	Protocol processing node
QOS	Quality of Service
RISC	Reduced Instruction Set Computers
ROPE	Reduced Operation Protocol Engine
RPC	Remote Procedure Call
RTL	Register Transfer Level
RTT	Round Trip Time

<i>SDTreq</i>	Session data transfer request primitive
<i>SDU</i>	Service data unit
<i>SM</i>	Shared Memory
<i>SNA</i>	Systems Network Architecture
<i>SPDU</i>	Session Protocol Data Unit
<i>SPE</i>	Software Performance Engineering
<i>SPS</i>	Standard Protocol Stack
<i>SRAM</i>	Static Random Access Memory
<i>SRC-REF</i>	Source Reference (field)
<i>SS</i>	Session Service
<i>TCP</i>	Transmission Control Protocol
<i>TLV</i>	Type-Length-Value
<i>TPDU</i>	Transport Protocol data unit
<i>TPDU-NR</i>	DT TPDU number (field)
<i>TSD</i>	Time Sequence Diagram
<i>UD</i>	User Data
<i>VHDL</i>	VHSIC Hardware Description Language
<i>VHSIC</i>	Very High Speed Integrated Circuit
<i>VLSI</i>	Very Large Scale Integration
<i>VMTP</i>	Versatile Message Transaction Protocol
<i>XTP</i>	Xpress Transfer Protocol
<i>YR-TU-NR</i>	Sequence Number response (field)

Bibliography

- [1] 1076-1987 Standard VHDL Language Reference Manual, The IEEE computer society Press 1991.
- [2] Abeyesundara B.W., Kamal A.E., "High-Speed Local Area Networks and Their Performance: A Survey," in ACM Computing Surveys, Vol. 23, No. 2, Jun 1991.
- [3] Anastasi G., Conti M., Gregori E., "TPR: A Transport Protocol for Real Time Services in an FDDI Environment," in Proc. IFIP Workshop on Protocols for High-Speed Networks, Zurich, May 9-11, 1990.
- [4] Andrews W., "High-performance networks challenge Ethernet" in Computer Design, July 1992.
- [5] Armstrong, J., "Chip Level Modelling in VHDL," Prentice Hall, 1988.
- [6] Arnold E. et al, "The design of NECTAR: A network backplane for heterogenous multicomputers," in proc. ASPLOS-III, Apr, 1989.
- [7] Arnold G., "Internet Protocol Implementation Experiences in PC-NFS," in ACM SIGCOMM Workshop on Frontiers in Computer Communications Technology, Vermont, Aug. 1987.
- [8] Bailey D.L., "Configuration Guide for UNISON/DY-4 134 Multiprocessor Testbed," Carleton University, 1989.
- [9] Bailey M.L., Pagels M.A., Peterson L.L., "The x-chip: An Experiment in Hardware Demultiplexing," Department of Computer Science, University of Arizona, 1993.
- [10] Baker I.E., "File Transfer Access and Management: FTAM ISO 8571," in Proceedings of Intl. Conference on OPEN Systems, London, Mar. 1987.
- [11] Balraj T.S. and Yemini Y., "Putting the Transport Layer on VLSI - the PROMPT protocol chip". IFIP, Stockholm, May 13-15, 1992.
- [12] Banks D. and Prudence M., "A High-Permanence Network Architecture for a PA-RISC Workstation," in IEEE Journal on Selected Areas in Communications, Vol. 11, No.2, February 1993.
- [13] Beach B., "UltraNet: An Architecture for Gigabit Networking," in Proc. 15th Conference on Local Computer Networks, Minnesota Oct, 1990.
- [14] Berenbaum A., Dixon J., Iyengar A., Keshav S., "A Flexible ATM-Host Interface for XUNET II," in IEEE Network, July 1993.
- [15] Bershad B.N., Anderson T.E., Lazowska E.D., Levy H.M., "Lightweight Remote Procedure Call," in ACM Trans. in Computer Systems, Vol. 8, No. 1, pp. 37-55, Feb. 1990.

- [16] Bever M., Schaffer U. and Schottmuller C., "ISO OSI FTAM and High-speed file transfer: No contradiction," in IFIP High Performance Networking IV, Elsevier Science Publishers B.V. North-Holland, 1993.
- [17] Bever M., Schaffer U., "Coding Rules for High Speed networks," in Upper Layer Protocols, Architectures and Applications. Elsevier Science Publishers B.V., 1992.
- [18] Bhandarkar D., Clark D.W., "Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization," in the 4th Int'l Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, California, Apr 8-11, 1991.
- [19] Biagioni E., Cooper E., Sansom R., "Designing a Practical ATM LAN" in IEEE Network, March 1993.
- [20] Bilgic M., Sarikaya B., "An ASN.1 Encoder/Decoder and Its Performance," in IFIP Protocol Specification Testing and Verification, Elsevier Science Publishers B.V., North-Holland, 1990.
- [21] Bjorkman M., Gunningberg P., "Locking Effects in Multiprocessor Implementations of Protocols" in ACM SIGCOMM 1993.
- [22] Black U., "OSI: A Model for Computer Communications Standard," Prentice-Hall Inc, 1991.
- [23] Blair G. et al, "A Network Interface Unit to support Continuous Media," in IEEE Journal on Selected Areas in Communications, vol. 11 No. 2 Feb 1993.
- [24] Bochmann G.V., "Specifications of a Simplified Transport Protocol Using Different Formal Description Techniques," Publication No. 623a, University of Montreal, Apr. 1987.
- [25] Bolot J.C., Shankar A.U., "Dynamical Behavior of Rate-Based Flow Control Mechanisms," in ACM SIGCOMM Computer Communications Review, Vol. 20, No. 2, pp. 35-49, Apr. 1990.
- [26] Bondi A.B., "Problems in expediting protocol processing: performance analysis of kernel mode priority scheduling and front end protocol processors," in Performance Evaluation 13, pp. 253-265, 1991.
- [27] Borman D.A., "Implementing TCP/IP on a Cray computer," in ACM SIGCOMM, Comp. Commun. Review, vol. 19, no. 2, pp. 11-15, Apr. 1989.
- [28] Bortolotto, E.P., "HP OSI Express Design for Performance," in Hewlett Packard Journal, Feb. 1990.
- [29] Bowen D., "Open distributed processing," in Computer Networks and ISDN systems 23, pp. 195-201, North-Holland, 1991.
- [30] Braden R., Borman D., Partridge C., "Computing the Internet Checksum," in ACM SIGCOMM, Comp. Commun. Review, vol. 19, no. 2, pp. 86-94, Apr. 1989.

- [31] Brandriff R.K., Lynch C.A., Needleman M.H., "Development of a TCP/IP for the IBM/370," in ACM SIGCOMM on Proceedings of the 9th Data communications Symp., B.C., Sep. 1985.
- [32] Braun T., Zitterbart M., "A Parallel Implementation of XTP on transputers," in Proc. 16th Conference on Local Computer Network, Oct. 1991.
- [33] Brown G.M., Gouda M.G., Miller R.E., "Block Acknowledgement: Redesigning the Window Protocol," in ACM SIGCOMM, Symp. on Communication Architecture and Protocols, pp. 128-135, Sep. 1989.
- [34] Byers, T. J., "Microprocessor communications support chips", 1987.
- [35] Caneschi F., "Hints for the Interpretation of the ISO Session Layer," in ACM SIGCOMM Comp. Commun. Review, vol. 16, no. 4, pp. 34-72, Aug. 1987.
- [36] Carter J.B., Zwaenepoel W., "Optimistic Implementation of Bulk Data Transfer Protocols," in Performance Evaluation Review, pp. 61-69, Vol. 17, No. 1, May 1989.
- [37] Cheriton D. and Williamson C., "VMTP as the transport layer for high-performance distributed systems," IEEE Commun. Mag., vol.27, pp. 37-44, June 1989.
- [38] Cheriton D., "VMTP: A Transport Protocol for the Next Generation of Communication System," in ACM SIGCOMM, Symp. on Communications Architecture and Protocols, Vermont, Aug. 1986.
- [39] Cheriton D., "VMTP: Versatile Message Transaction Protocol: Protocol Specification," RFC 1045, Feb. 1988.
- [40] Cheriton D., Zwaenepoel W., "Distributed Process Groups in the V kernel," in ACM trans. on Computer Systems, Vol. 3, No. 2, pp. 77-107, May 1985.
- [41] Chesson G., "XTP/PE Design Considerations," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, pp. 27-33, 1989.
- [42] Chong H.Y., "Software Development and Implementation of NBS Class-4 Transport Protocol," in Computer Networks and ISDN systems 11, pp. 353-365, North-Holland, 1986.
- [43] Clark D. and Tennenhouse D., "Architectural Considerations for a New Generation of Protocols," in ACM SIGCOMM 1990.
- [44] Clark D., Jacobson V., Romkey J., and Salwen H., "An analysis of TCP processing overhead," in IEEE Commun. Mag., vol. 27, pp. 23-29, June 1989.
- [45] Clark D., Lambert M., and Lixia Zhang, "NETBLT: A Bulk Data Transfer Protocol," RFC 998, Mar. 1987.
- [46] Clark D., Lambert M., and Lixia Zhang, "NETBLT: A high throughput Transport protocol," in ACM SIGCOMM '87 Workshop Comp Commun Review, vol. 17, no. 5, Special Issue, pp. 353-359, Aug 1987.

- [47] Clark D., "Modularity and Efficiency in Protocol Implementation," RFC 817 1982.
- [48] Clark D., Shenker S. and Zhang L., "Supporting Real-time Applications in an Integrated Services Packet Network: Architecture and Mechanism," in ACM SIGCOMM'92.
- [49] Clark D., "The Structuring of Systems Using UPCALLS," in ACM, 1985.
- [50] Clark D., "Window and Acknowledgment Strategy in TCP," RFC 813, Jul. 1982.
- [51] Cockcroft A., "Performance Tuning of SPARCstations", in SunUSER, Birmingham, UK, 1991.
- [52] Coelho D.R., "The VHDL Handbook," Kluwer Academic Publishers, 1989.
- [53] Colella R., Aronoff R., Mills K., "Performance Improvements for ISO Transport," in ACM SIGCOMM on 9th Data communications Symp. 1985.
- [54] Comer D.E., "Internetworking with TCP/IP: Vol. 1 Principles, Protocols and Architecture," Englewood Cliffs, NJ: Prentice Hall, 1991.
- [55] Comerford R., Watson G.F., "Memory Catches Up," in IEEE Spectrum, Oct 1992.
- [56] Cooper E.C, Menzilcioglu O., Sansom R. and Bitz F., "Host Interface Design for ATM LANs," in proc. 16th Conference on Local Computer Networks, Oct. 1991.
- [57] Cooper E.C, Steenkiste P.A., Sansom R.D. and Zill B.D., "Protocol Implementation on the Nectar Communication Processor," in ACM SIGCOMM'90, 1990.
- [58] Crowcroft J., Wakeman I., Wang Z., Sirovica D., "Is Layering Harmful?," in IEEE Network Magazine, pp. 20-24, Jan. 1992.
- [59] Dalton C., Watson G., Banks D., Calamvokis C., Edwards A. and Lumley J., "Afterburner," in IEEE Network July 1993.
- [60] Davie B.S., "Architecture and Implementation of a High-Speed host Interface," in IEEE Journal on selected areas in communications, Vol. 11, No. 2, February 1993.
- [61] DeSchon A.L., "A Survey of Data Representation standards," RFC 971, 1986.
- [62] Despres R., "Global high speed WAN architecture for the 90's," in Computer Networks and ISDN systems 23, pp. 125-128, North-Holland, 1991.
- [63] Diot C., Roca V., "XTP/KRM Implementation on a transputer Network," in Proc. 16th Conference on Local Computer Network, Oct. 1991.
- [64] Doeringer W.A., Dykeman D., Kaiserwerth M., Meister B., Rudin H., and Williamson R., "A survey of Light-Weight Transport protocols for High Speed Networks," in IEEE Trans. on Comm., vol. 38, No. 11, pp 2025-2039, Nov. 1990.
- [65] Doshi B.T., Johri P.K., Netravali A.N and Sabnani K.K. "Error and Flow Control Performance of a High Speed Protocol," in IEEE trans on communications, vol. 41, No. 5, May 1993.

- [66] Druschel P., Abbott M.B., Pagels M.A. and Peterson L.L., "Network Subsystem Design," in IEEE Network July 1993.
- [67] Druschel P., Peterson L.L., "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility", in the Proceedings of the 14th ACM Symposium on Operating Systems Principles, Dec 1993.
- [68] Eager D.L., Zahorjan J., Lazowska E.D., "Speedup versus Efficiency in Parallel Systems," in IEEE Trans. on Computers, pp. 408-423, Vol. 38, No. 3, Mar 1989.
- [69] Eldridge C.A., "Rate Controls in Standard Transport Layer Protocols," in ACM SIGCOMM Computer Communication Review, Vol 22, No. 3, July 1992.
- [70] Emmons W.F., Chandler A.S., "OSI Session Layer: Services and Protocols," in Proceedings of the IEEE, pp. 1397-1400, Vol.71, No.12, Dec. 1983.
- [71] Farmwald et al, "Special Report/Memory". in IEEE Spectrum, Oct 1992.
- [72] Feldmeier D.C., Biersack E.W., "Comparison of Error Control Protocols for High Bandwidth-Delay Product Networks," in Proc. IFIP Workshop on Protocols for High-Speed Networks, Zurich, May 9-11, 1990.
- [73] Feldmeier D.C., "Multiplexing Issues in Communications System Design," in ACM SIGCOMM 1990.
- [74] Ferrari D. and Verma D.C., "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," in IEEE Journal on Selected Areas in COmmunications, vol. 8, no. 4, April 1990.
- [75] Fletcher J.G., "An Arithmetic Checksum for Serial Transmissions" in IEEE Transactions on Communications, Vol. Com-30, No. 1, Jan 1982.
- [76] Fluckiger F., "From megabit to gigabit: possible transition scenarios," in Computer Networks and ISDN systems 23, pp. 129-138, North-Holland, 1991.
- [77] Folts H., "OPEN Systems Standards", in IEEE Network, Vol.2, No. 3, May 1988.
- [78] Fong K., Reinstedler J., "The Development of an OSI Application Layer Protocol Interface," in ACM SIGCOMM Computer Communications Review, Vol. 19, No. 3, pp. 21-57, July. 1989.
- [79] Franks R.G., "Experience in Concurrency in OSI Communications Software: A comparative Performance Study," Masters Thesis, Carleton University, 1989.
- [80] Gall D.L., "MPEG," in Communications of the ACM, Apr. 1991.
- [81] Giarrizzo D., Kaiserswerth M., Wicki T. and Williamson R., "High-Speed Parallel Protocol Implementation," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1989.

- [82] Goldberg M., Neufeld G. and Ito M., "A parallel approach to OSI connection-oriented protocols", in Proc. of the 3rd IFIP International Workshop on Protocols for High-Speed Networks, Stockholm, May 13-15, 1992.
- [83] Gunningberg P., Pink S., Bjorkman M., Sjodin P., Nordmark E., Stromquist J., "Application Protocols and Performance Benchmarks," in IEEE Communications Magazine, Jun 1989.
- [84] Haas Z., "A Communication Architecture for High-Speed Networking," in IEEE INFOCOM, pp. 433-441, June 1990.
- [85] Haas Z., "A Protocol Structure for High-Speed Communication over Broadband ISDN," in IEEE Network Magazine, pp. 64-70, Jan. 1991.
- [86] Harvey J.D. and Weaver A.C., "Experience with the Abstract Syntax Notation One and the Basic Encoding Rules," in Proc. 16th Conference on Local Computer Networks, Oct. 1991.
- [87] Hennessy J.L and D.A. Patterson, "Computer Architecture: A quantitative approach," Palo Alto, California, Morgan Kaufmann Publishers 1991.
- [88] Henshall J., Shaw S., "OSI Explained - end to end computer communication standards," Ellis Horwood Limited, 1988.
- [89] Hockney R.W. and Jesshope C.R., "Parallel Computers 2," Adam Higler, 1988.
- [90] Huang J.H. and Chen C., "On Performance Measurements of TCP/IP and its device driver," in 17th Conference on Local Computer Networks, 1992.
- [91] Huitema C, Dabbous W., "Minimal Complexity for the Simplext Protocol," in IEEE INFOCOM'91, Conference on Computer Communications Bal Harbour, Fl, USA, March 1991.
- [92] Huitema C., Doghri A., "A High Speed Approach for the OSI Presentation Protocol," in Proc. IFIP Workshop Protocols for High-Speed Networks, 1989.
- [93] Huitema C., Doghri A., "Defining faster transfer syntaxes for the OSI Presentation Protocol," in ACM SIGCOMM Computer Communications Review, Vol. 19, No. 5, pp. 44-55, Oct. 1989.
- [94] Hutchinson N.C., Peterson L.L., Abbot M.B., O'Malley S., "RPC in the x-kernel: Evaluating New Design Techniques," in Proceedings of the 12th Symp. on Operating System Principles, Dec. 1989.
- [95] Hutchinson N.C., Peterson L.L., Rao H., "The x-Kernel: An Open Operating System Design," in Proceedings of the 2nd workshop on workstation operating system, California, Sept 1989.
- [96] Hwang K. and Briggs F.A., "Computer Architecture and Parallel Processing," McGraw-Hill, 1989.

- [97] Information processing systems — OSI Basic connection oriented session protocol specification, standard ISO-8327, 1987.
- [98] Information processing systems — OSI Basic connection oriented session service definition, standard ISO-8326, 1987.
- [99] Information processing systems — OSI Connection oriented presentation protocol specification, standard ISO-8823, 1988.
- [100] Information processing systems — OSI Connection oriented presentation service definition, standard ISO-8822, 1988.
- [101] Information processing systems — OSI Connection oriented transport protocol specification, standard ISO-8073, 1986.
- [102] Information processing systems — OSI Specification of Abstract Syntax Notation One (ASN.1), standard ISO-8824, 1987.
- [103] Information processing systems — OSI Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), standard ISO-8825, 1987.
- [104] Information processing systems — OSI transport service definition, standard ISO-8072, 1986.
- [105] Information processing systems — OSI Basic Reference Model, standard ISO-7498, 1984.
- [106] Ito M.R., Takeuchi L.Y., Neufeld G.W., "A Multiprocessor Approach for Meeting the Processing Requirements for OSI," in IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, Feb 1993.
- [107] Jacobson V., "4BSD TCP Header Prediction," in ACM SIGCOMM, Comp. Commun. Review, vol. 20, no. 2, pp. 13-16, Apr. 1990.
- [108] Jacobson V., "Efficient Protocol Implementation," ACM SIGCOMM'90 tutorial, Sept 1990.
- [109] Jacobson V., Notes on TCP/IP mailing list, tcp-ip@SRI-NIC.ARPA, Mar. 1988.
- [110] Jain B.N., Agrawala A.K., "Open Systems Interconnection: Its Architecture and Protocols," Elsevier Science Publishing Company, 1990.
- [111] Jain N., Schwartz M. and Bashkow T.R., "Transport Protocol Processing at Gbps rates," ACM SIGCOMM '90 Symp, Philadelphia, pp. 188-199, Sep. 24-27, 1990.
- [112] Jain R., "The Art of Computer Systems Performance Analysis," John Wiley and Sons, 1991.
- [113] Johnston C.A., Chao H.J., "The ATM Layer Chip: An ASIC for B-ISDN Applications," in IEEE Journal on Selected Areas in Communications, vol. 9, no. 5, pp. 741-750, Jun 1991.

- [114] Kanakia H. and Cheriton D., "The VMP network adapter board (NAB): High-performance network communication for multiprocessors," ACM SIGCOMM '88 Symp., Stanford, CA, pp. 175-187, Aug. 16-19, 1988.
- [115] Karn P., Partridge C., "Improving Round Trip Time Estimates in Reliable Transport Protocols," in ACM SIGCOMM Workshop on Frontiers in Computer Communications Technology, Vermont, Aug. 1987.
- [116] Kay J., Pasquale J., "The Importance of Non-Data Touching Processing Overheads in TCP/IP" in ACM SIGCOMM'93, 1993.
- [117] Kent C.A., Mogul J.C., "Fragmentation Considered Harmful," ACM SIGCOMM '87 Workshop Comp Commun Review, vol. 17, no. 5, Special Issue, Aug 1987.
- [118] Khrishnakumar A.S., Sabnani K., "VLSI Implementations of Communication Protocols — A Survey," IEEE Journal on Selected Areas in Communications, vol. 7, no. 7, pp. 1082-1090, Sep 1989.
- [119] Koelman W.A., "Implementing Lower Layer Open Systems Interconnection Protocols in the UNIX Streams Environment," Technical Note 384, Shape Technical Centre (NATO), The Hague, Jul. 1991.
- [120] Koufopavlou O.G., Tantawy A.N., Zitterbart M., "Analysis of TCP/IP for High Performance Parallel Implementations". IEEE 17th Conference on Local Computer Network, 1992..
- [121] Krishnakumar A.S., Krishnamurthy B., Sabnani K., "Translation of Formal Protocol Specifications to VLSI Designs," in Protocol Specification, Testing and Verification, VII. Elsevier Science Publishers B.V., IFIP, 1987.
- [122] Lambert M., "On testing the NETBLT Protocol over Divers Networks," RFC 1030, Nov. 1987.
- [123] Leiner B., "Critical Issues in High Bandwidth Networking," RFC 1077, 1988.
- [124] Leiner B.M., Cole R., Postel J, and Mills D., "The DARPA Internet protocol suite," in IEEE Commun. Mag., vol. 23, Mar. 1985.
- [125] Leung S., Shanblatt M.A., "ASIC System Design with VHDL: A Paradigm," Kluwer Academic Publishers, 1989.
- [126] Linington P.F., "Why OSI?," Computer Networks and ISDN systems 17, pp. 287-290, North-Holland, 1989.
- [127] Lipsett, Schaeffer and Ussery, "An Introduction to VHDL: Hardware Description and Design," Kluwer Academic Publishers, 1989.
- [128] Little T.D.C. and Ghafoor A., "Network Considerations for Distributed Multimedia Object Composition and Communication," in IEEE Network Magazine, Nov 1990.

- [129] Little T.D.C., Ghafoor A., "Synchronization and Storage Models for Multimedia Objects", in *IEEE Journal on Selected Areas in Communications*, Vol 8 No. 3 April 1990.
- [130] Lyles J.B., Swinehart D.C., "The Emerging Gigabit Environment and the Role of Local ATM," in *IEEE Communications Magazine*, Apr 1992.
- [131] Maclean R.A., Barvick S.E., "An Outboard Processor For High Performance Implementation of Transport Layer Protocols," *GLOBECOM*, Arizona, Dec. 1991.
- [132] Maly K. et al "Parallel TCP/IP for Multiprocessor Workstations," in *IFIP High Performance Networking IV*, Elsevier Science Publishers B.V. North-Holland, 1993.
- [133] Marathe M.V., Smith R.A., "Performance of a MAP Network Adapter," in *IEEE Network*, pp. 82-89, vol.2, No.3, May 1988.
- [134] McGuffin L.J., Reid L.O., Sparks S.R., "MAP/TOP In CIM Distributed Computing," *IEEE Network*, pp. 23-31, vol.2, no. 3, May. 1988.
- [135] Meister B., Janson P., Svobodova L., "File Transfer In Local Area Networks: A Performance Study," *IEEE* 1985.
- [136] Metcalfe R.M., "Computer/Network Interface Design: Lessons from Arpanet and Ethernet," in *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 2, Feb 1993.
- [137] Mills D.L., Davis M., "Internet Architecture Workshop: Future of the Internet System Architecture and TCP/IP Protocols," in *ACM SIGCOMM Computer Communications Review*, Vol. 20, No. 1, Jan. 1990.
- [138] Mogul J.C., Borg A., "The Effect of Context Switches on Cache Performance," in *4th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, ACM 1991.
- [139] Multiprocessor Toolsmiths Incorporated. *Unison Real-Time Multitasking, Multiprocessing Operating System User's guide*, 1987.
- [140] Neilson J.E., *A PARASOL Tutorial*, School of Computer Science, Carleton University, Jan. 1992.
- [141] Neilson J.E., *PARASOL Users' Manual*, School of Computer Science, Carleton University, Jan. 1992.
- [142] O'Malley S.W. and Peterson L.L., "A Dynamic Network Architecture," in *ACM Transactions on Computer Systems* Vol 10, No. 2 May 1992.
- [143] Onions J., "ISODE: In Support of Migration," in *Computer Networks and ISDN systems* 17, pp. 362-366, North-Holland, 1989.
- [144] Papadopoulos C., Parulkar G.M., "Experimental Evaluation of SUNOS IPC and TCP/IP Protocol Implementation", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 2, Apr 1993.

- [145] Partridge C., "How Slow Is One Gigabit Per Second?," in ACM SIGCOMM, Comp. Commun. Review, vol. 20, no. 1, Jan. 1990.
- [146] Partridge C., "Protocols for High-speed networks: Some questions and a few answers," in Computer Networks and ISDN Systems 25, North-Holland 1993.
- [147] Parulkar G.M., Turner J.S., "Towards a Framework for High-Speed Communication in a Heterogeneous Networking Environment," in IEEE Network Magazine, pp. 19-27, Mar. 1990.
- [148] Perry D.L., "VHDL," McGraw-Hill Inc., 1991.
- [149] Peterson L.L., Hutchinson N.C., O'Malley S.W., Rao H., "The x-Kernel: A Platform for Accessing Internet Resources," Oct. 1989.
- [150] Popescu-Zeletin R., "Implementing the ISO-OSI Reference Model," in ACM SIGCOMM, 8th Data Communications Symp., North Falmouth, Oct. 1983.
- [151] Porta T.F.L., Schwartz M., "Architectures, Features, and Implementation of High-Speed Transport Protocols," IEEE Network Magazine, pp. 14-22, May 1991.
- [152] Porta T.F.L., Schwartz M., "Architectures, Features, and Implementation of High-Speed Transport Protocols," in GLOBECOM, Arizona, Dec. 1991.
- [153] Protocol Engines Inc., "XTP Protocol Definition," version 3.3, Dec. 1988.
- [154] Rainer H., Munfred N., "Integrated Broadband Networks: An Introduction to ATM based networks," in Addison Wesley, 1991.
- [155] Ramakrishnan K.K., "Performance Issues in Designing Network Interfaces: A case study," in IFIP High Performance Networking IV, Elsevier Science Publishers B.V. North-Holland, 1993.
- [156] Ramanathan S. and Rangan P.V., "Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems", in The Computer Journal, Vol. 36 No. 1, Jan 1993.
- [157] Rashid R., Baron R., Forin A., Golub D., Jones M., Julin D., Orr D., Sanzi R. "Mach: A Foundation for Open Systems," in Proc. of the 2nd workshop on Workstation OS, California, Sept 1989.
- [158] Reijer H., "IMPLOSIE: Implementation of an OSI environment," in Proceedings of Intl. Conference on OPEN Systems, London, Mar. 1987.
- [159] Rose M.T., "The Open Book: A Practical Perspective on OSI," Prentice-Hall Inc., 1990.
- [160] Ross F.E., "An Overview of FDDI: The Fiber Distributed Data Interface," in IEEE Journal on Selected Areas in Communications, vol. 7, no. 7, pp. 1043-1051, Sep 1989.
- [161] Rudin H., Williamson R., "Faster, More Efficient Streamlined Protocols," IEEE Communications Magazine, Jun 1989.

- [162] Rutsche E., Kaiserwerth M., "TCP/IP on the Parallel Protocol Engine," in IFIP High Performance Networking IV, Elsevier Science Publishers B.V. (North Holland), 1993.
- [163] Sanders R.M., Weaver A.C., "The Xpress Transfer Protocol (XTP) - A Tutorial," in ACM SIGCOMM Computer Communications Review, Vol. 20, No. 5, Oct. 1990.
- [164] Schmidt D.C., Suda T., "Transport System Architecture Services for High- Performance Communications Systems". IEEE Journal on Selected Areas in Communications, Vol. 11, No. 4, May 1993.
- [165] Schwaderer W.D., "XTP in VLSI Protocol Decomposition for ASIC Implementation," in proc 15th Conference on Local Computer Network, 1990.
- [166] Schwartz M., "Telecommunications Networks: Protocols, Modelling and Analysis." Addison-Wesley Publishing Company, 1988.
- [167] Shepphard G.B., "Implementation of an OSI environment," in Proceedings of Intl. Conference on OPEN Sytems, London, Mar. 1987.
- [168] Sidenius M.A., "Hardware Support for Implementation of Transport Layer Protocols," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1990.
- [169] Siegel M., Williams M. and Robler G., "Overcoming Bottlenecks in High-Speed Transport Systems," in Proc of 16th Conference on Local Computer Networks, Oct. 1991.
- [170] Sklower K., "Improving the Efficiency of the OSI Checksum Calculation," in ACM SIGCOMM Computer Communications Review, Vol. 19, No. 5, pp. 32-43, Oct. 1989.
- [171] Smith C.U., "Applying synthesis principles to create responsive software systems," IEEE Trans. on Software Engineering, 1988.
- [172] Smith C.U., "Independent General Principles for Constructing Responsive Software Systems," in ACM Trans. on Computer Systems, pp. 1-31, Vol. 4, No. 1, Feb. 1986.
- [173] Smith C.U., "Performance Engineering of Software Systems," The S.E.I. series in software engineering, 1990.
- [174] Smith J.M. and Traw C.B.S., "Giving Applications Access to Gb/s Networking," in IEEE Network July 1993.
- [175] Stallings W., "Data and Computer Communications," Macmillan Publishing Company, Second Edition, 1988.
- [176] Stallings W., "Handbook of Computer Communications Standards, Vol. 1, The Open Systems Interconnection (OSI) Model and OSI Related Standards," Macmillan Publishing Company, 1989.
- [177] Stallings W., "Handbook of Computer Communications Standards, Vol. 3 DoD Protocol Standards," Macmillan Publishing Company, 1989.

- [178] Steedman D., "ASN.1 - The Tutorial and Reference," Technology Appraisals Ltd., 1990.
- [179] Sterbenz J.P.G. and Parulkar G.M., "AXON: A High Speed Communications Architecture for Distributed Applications," in IEEE INFOCOM, Jun 1990.
- [180] Sterbenz J.P.G. and Parulkar G.M., "AXON Host-Network Interface architecture for Gigabit communications," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1990.
- [181] Strayer W.T., Dempsey B.J. and Weaver A.C., "XTP: The Xpress Transfer Protocol," by Addison-Wesley Publishing, 1992.
- [182] Strayer W.T., Weaver A.C., "Performance Measurement of Data Transfer Services in MAP," in IEEE Network, pp. 75-81, vol.2, No.3, May 1988.
- [183] Svobodova L., "Implementing OSI Systems," in IEEE Journal on Selected Areas in Communications, vol. 7, no. 7, pp. 1115-1130, Sep 1989.
- [184] Svobodova L., Janson P.A., Mumprecht E., "Heterogeneity and OSI," in IEEE Journal on Selected Areas in Communications, vol. 8, no. 1, pp. 67-79, Jan 1990.
- [185] Svobodova L., "Measured Performance of Transport Service in LANs," in Computer Networks and ISDN systems 18, pp. 31-45, North-Holland, 1989.
- [186] SYNOPSIS Design AnalyzerTM Reference Manual, Version 2.0, May 1991.
- [187] SYNOPSIS VHDL Compiler Reference Manual, Version 2.0, May 1991.
- [188] Tanenbaum A.S., "Computer Networks," Prentice Hall, 2nd Edition, 1988.
- [189] Tawbi W., Dupuy S., Horlait E., "High Speed Protocols: State of the Art in Multimedia Applications," in Information Network and Data Communication, IV Elsevier Science Publishers B.V. IFIP, 1992.
- [190] Texas Instrument TMS320C40 Source Debugger Manual, 1992.
- [191] Texas Instrument TMS320C40 User's Guide, 1991.
- [192] TGB 1000 Series 0.8um BiCMOS Gate Arrays Macro Library Summary, Application Specific Integrated Circuits, Texas Instruments, Dec. 1991.
- [193] Thekkath C.A., Nguyen T.D., Moy E., Lazowska E.D., "Implementing Network Protocols at User Level," in ACM SIGCOMM'93, 1993.
- [194] Thia Y.H., Woodside C.M., "A Parallel Optimistic Bypass Architecture (POBA) for high-speed bulk data protocol processing", submitted to the IEEE/ACM Trans on Networking for publication, 1993.
- [195] Thia Y.H., Woodside C.M., "A Reduced Operation Protocol Engine (ROPE) for a multiple-layer bypass system", to be submitted to the 4th IFIP International Workshop on Protocols for High-Speed Networks for publication, 1993.

- [196] Thia Y.H., Woodside C.M., "High-Speed Protocol Bypass Algorithm with Window Flow Control", in Proc. of the 3rd IFIP International Workshop on Protocols for High-Speed Networks, Stockholm, May 13-15, 1992.
- [197] Topolovic C. (Editor), "Experimental Internet Stream Protocol, Version 2 (ST-II)," RFC 1190, Oct 1990.
- [198] Transmission Control Protocol, RFC 793, 1981.
- [199] Varghese G. and Lauck T., "Hashed and Hierarchical Timing Wheels: Data structures for the efficient implementation of a Timer facility," in Proc. of the 11th ACM Symp. on Operating System Principles, Nov. 1987.
- [200] Wahl T., Salmony M., "Optimisation of Transport Protocols for High-Speed," in Information Network and Data Communication, IV. Elsevier Science Publishers B.V., Jan 1992.
- [201] Watson R.W. and Mamrak S.M., "Gaining efficiency in smth transport services by appropriate design and implementation choices," in ACM trans. on Computer Systems, vol. 5, no. 2, pp. 97-120, May 1987.
- [202] Watson R.W., "The delta-t transport protocol: Features and experience Useful for High Performance Networks," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, pp.3-17, 1989.
- [203] White J.E., "ASN.1 and ROS: The Impact of X.400 on OSI," in IEEE Journal on Selected Areas in Communications, vol. 7, no. 7, pp. 1060-1072, Sep 1989.
- [204] Wilson J.T., "A Prototyping Environment for Concurrent Systems," Dept of Computer Science, Carleton University, Aug 1990.
- [205] Woodside C.M. and Montealegre J.R., "The effect of buffering strategies on protocol execution performance," in IEEE Trans. Commun., vol. COM-37, pp.545-554, June 1989.
- [206] Woodside C.M., Franks G., Constantin R., "A Performance Model for Protocol Execution," Canadian Conf. on Elect. and Computer Engineering, Vancouver, B.C. Canada, pp. 296-300, Nov. 1988.
- [207] Woodside C.M., Franks R.G., "Alternative Software Architecture for Parallel Protocol Execution with Synchronous IPC," in IEEE/ACM Trans. on Networking, vol. 1, no. 2, 1993.
- [208] Woodside C.M., Li Y., "Performance Petri Net Analysis of Communications Protocol Software by Delay-Equivalent Aggregation," in Proc. of the fourth Intl. Workshop of Petri Nets and Performance Models, Australia, Dec. 1991.
- [209] Woodside C.M., Montealegre R., Buhr R.J.A., "A Performance Model for Hardware/Software Issues in Computer-Aided Design of Protocol Systems," in ACM

- SIGCOMM, Tutorial and Symp. on Communications Architecture and Protocols, Canada, Jun 1984.
- [210] Woodside C.M., Ravindran K. and Franks R.G., "The protocol bypass Concept for High Speed OSI Data transfer," in Proc. IFIP Workshop on Protocols for High-Speed Networks, Zurich, May 9-11, 1990.
 - [211] Zhang L., "Why TCP Timers Don't Work Well," in ACM SIGCOMM, Symp. on Communications Architecture and Protocols, Aug. 1986.
 - [212] Zimmermann H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection". IEEE Transactions on communications, vol. Com-28, No.4, April 1980..
 - [213] Zissopoulos A., Tropper C., "On Buffer Allocation in Transport Protocols," in Computer Networks and ISDN systems 16, pp. 383-393, North-Holland, 1989.
 - [214] Zitterbart M., "A Multiprocessor Architecture for High Speed Network Interconnections," in Proc. IEEE INFOCOM'89, Ottawa, Apr. 1989.
 - [215] Zitterbart M., "High-Speed Protocol Implementations based on Multiprocessor-Architecture," in Proc. IFIP Workshop Protocols for High-Speed Networks, Zurich, May 9-11, 1989.
 - [216] Zitterbart M., "High-Speed Transport Components," in IEEE Network Magazine, pp. 54-63, Jan. 1991.
 - [217] Zitterbart M., "Parallel Protocol Implementations For High Speed Networks," in IEEE Intl. Telecommunications Symp. Brazil, Sep 1990.
 - [218] Zitterbart M., Stiller B. and Tantawy A. "A Model for Flexible High-Performance Communications Subsystems," in IEEE Journal on Selected Areas in Communications, vol 11, No. 4, May 1993.

Appendix A Pseudo Code for bypass

The pseudo 'C' code algorithm for bypassing with window flow control for the send and receive path are presented below. The first describes the Send bypass test for DT PDU and the send bypass stack. Secondly, the Receive bypass test for DT PDU and the receive bypass stack is presented. Lastly, the algorithm for bypassing AK TPDU is given.

```
/*
 * PSEUDO CODE of Sender algorithm for window flow control.
 * Bypass SESSION and TRANSPORT(TP2) layers during data transfer phase
 * Bypass Test conducted just prior to protocol processing of
 * SESSION layer. Message here refers to Session layer Users' PDU.
 */
IF (Layer == SESSION &&
    Service_primitive == SDTreq &&
    (Sizeof(Message) + Header_size <= TPDU_SIZE)) THEN
    BEGIN
        IF (transmit_bypass_flag == TRUE) THEN
            execute_send_bypass_procedure;
        ELSE IF (Session_Layer_State == STA713 &&
            Transport_Layer_State == OPEN &&
            Session_Entity_Empty &&
            Transport_Entity_Empty) THEN
            BEGIN
                transmit_bypass_flag = TRUE;
                execute_send_bypass_procedure;
            END
        ELSE
            BEGIN
                /* Leave Data unit in input queue and Flush SPS */
                Process_PDUs_in_SPS;
            END
        END
    ELSE
        BEGIN
            /* Send Bypass Not Possible */
            Usual_Processing_Path;
        END
    END
```

```

execute_send_bypass_procedure()
BEGIN
    /* Send Bypass possible */
    IF (X_NR == X_YR_TU_NR + X_CDT) THEN
        BEGIN
            /* Credit exhausted and hence Flow stopped.
            Queuing for Send Bypass */
            Start_next_layer;
        END
    ELSE
        BEGIN
            /* Credit Available, hence process message to send */
            TPDU_NR_field = X_NR;
            EOT_field = 1;
            X_NR = X_NR + 1;
            Generate_Transport_PDU_and_Interface_control_information;
            Enqueue_Buffer_To_Transport_Layer_Provider;
            Start_Next_Layer;
        END
    END
END

```

STA713 -- Data transfer state of OSI session layer

```

/*
* PSEUDO CODE of Receiver algorithm for window flow control.
* Bypass SESSION and TRANSPORT(TP2) layers during data transfer phase
* Bypass Test conducted just prior to protocol processing of the
* TRANSPORT layer.
*/
IF (Layer == TRANSPORT &&
    Match_Header) THEN
    BEGIN
        IF (receive_bypass_flag =- TRUE) THEN
            execute_receive_bypass_procedure;
        ELSE IF (Transport_Layer_State == OPEN &&
            Session_Layer_State == STA713 &&
            Transport_Entity_Empty &&
            Session_Entity_Empty) THEN
            BEGIN
                receive_bypass_flag = TRUE;
                execute_receive_bypass_procedure;
            END
        ELSE
            BEGIN
                /* Leave Data unit in queue and flush SPS */
                Process_PDUs_in_SPS;
            END
        ELSE

```

```

        BEGIN
            /* Receive Bypass Not Possible */
            Usual_Processing_Path;
        END

execute_receive_bypass_procedure()
BEGIN
    connection = DST_REF_field_of_transport_header;
    /* Receive Bypass possible */
    IF (Valid_PDU) THEN
        BEGIN
            /* Update Receive NR */
            R_NR = R_NR + 1;
            IF (Need_to_send_ACK?()) THEN
                BEGIN
                    /* Assemble ACK packet and enqueue */
                    R_YR_TU_NR = R_NR;
                    Allocate_CDT;
                    Generate_ACK_packet;
                    Enqueue_Buffer_To_transport_layer_provider;
                END
                Delete_transport_and_session_headers;
                Add_session_service_primitive;
                Enqueue_Buffer_To_session_layer_user;
            END
        ELSE
            BEGIN
                /* Invalid PDU error processing */
                Error_processing_of_invalid_PDU;
            END
        END
    END

/*
* PSEUDO CODE of Receiver algorithm for bypass reception of an ACK
* TPDU. Bypass Test conducted just prior to protocol processing of
* TRANSPORT (TP2) layer.
*/

    IF (Layer == TRANSPORT &&
        Match_ACK_Header)
        BEGIN
            IF (Transport_Layer_State == OPEN &&
                Transport_Entity_Empty)
                BEGIN
                    connection = DST_REF_field;
                    /* Receive ACK Bypass possible */
                    /* Update Window */

```



```

        X_CREDIT = CDT_field_of_ACK_TPDU;
        X_YR_TU_NR = YR_TU_NR_field_of_ACK_TPDU;
        Release_Buffer;
    END
ELSE
    BEGIN
        /* Leave Data Unit in queue and flush SPS */
        Process_PDUs_in_SPS;
    END
END
ELSE
    BEGIN
        /* Receive ACK Bypass Not Possible */
        Usual_Processing_Path;
    END
END

```

Appendix B TP4 Checksum

The checksum procedure enables the Transport entities to detect errors in communicating TPDU's with a high probability. Conceptually, this procedure is similar to the procedure employed at the data link layer to detect transmission errors. The algorithm used to compute the checksum parameter is however, different. The implementation as well as use of this procedure in Class 4 protocol is optional. The use of the procedure is negotiated at the time of connection establishment.

The algorithm states that for a sequence of bytes b_1 through b_n , the following iterative calculation,

$$C_{0,k+1} = C_{0,k} + b_k$$

$$C_{1,k+1} = C_{1,k} + C_{0,k+1}$$

with initial conditions $C_{0,0} = C_{1,0} = 0$ is computed. It can be easily shown to have a closed form expressed as:

$$C_0 = \sum_{i=1}^n b_i$$

$$C_1 = \sum_{i=1}^n (n + 1 - i)b_i$$

Fletcher analyzes the protection provided by these quantities (which he calls "check-bytes") for detecting errors under quite general circumstances, for bytes consisting of K bits (not just eight), and in fact considers higher order quantities. The ISO committees essentially adopted the case $K = 8$ and the two check-bytes C_0 and C_1 as providing the basis for their checksum. More specifically, they allow space in the packets for two contiguous bytes to be chosen so that when the two quantities C_0 and C_1 are computed in 8-bit, one's complement arithmetic for the packet as a whole, both sums are 0. With,

n — position of the first octet of the checksum parameter

L — length of the complete TPDU

X — value of the first octet of the checksum parameter

Y — value of the second octet of the checksum parameter

We can calculate X and Y such that

$$X = -C_1 + (L - n).C_0$$

$$\text{and } Y = C_1 - (L - n + 1).C_0$$

Appendix C ASN.1 Basic Encoding Rule for Integers

The advantage of the Abstract Syntax Notation One (ASN.1) with its corresponding encoding rules is that application designers do not need to be familiar with their details. This is analogous with the way that programmers using high-level languages do not have to know in detail how data structures are held in memory. The ASN.1 Basic Encoding Rules (BER) generate encoding which are a class known as type-length-value (TLV), so called because the basis of encoding is a structure made up of those three parts. With BER, the encoding of every data value in an abstract syntax, whether an entire PDU or some component of it, is constructed in the TLV style. The three parts are actually termed identifier (I), length (L) and contents (C).

The identifier conveys three pieces of information: the tag class of the data value being conveyed; the tag number; the form of the encoding — whether it is primitive or constructed.

The length (together with the form) allows the end of the contents to be found. The receiving system need not understand the tag to find the end of the contents, and this allows encoding to be skipped if it cannot (yet) be decoded.

The contents is the substance of the encoding, conveying the actual value. When the form of the encoding is primitive, the contents is simply a series of octets (zero or more):

When the form is constructed, the contents is a series of nested encoding, each having identifier, length and contents, as in:

The identifier is the first part of every encoding and occupies an integral number of octets, at least one. In the single octet variant, the two bit field CC represents the tag

bit 8 7	TAG CLASS
0 0	Universal
0 1	Application-wide
1 0	Context-specific
1 1	Private-use

bit 6	FORM
0	Primitive
1	Constructed

class, the single F bit the form, and the five field TTTTT the tag number.

The tag classes and form are represented as follows:

The length is the second part of every encoding and occupies an integral number of octets, at least one. The short variant can be used for any encoding whose contents is 127 octets or less in length, and is as follows:

where the field LLLLLLL represents, in unsigned binary, the length of the contents. Finally, every encoding has a content field.

This appendix describes only briefly the basic encoding rules and in particular the

encoding of an integer. A more comprehensive description is given in the ISO-8825 documentation. The encoding of an integer is primitive, and the contents octets represent the value using a two's-complement binary representation. In such a representation, where m bits are available to represent a value, the most significant bit is given a weight of $-(2^{m-1})$. (Lower numbered octets and higher numbered bits are more significant). All other bits have a positive weight, ranging from 2^{m-2} for the second most significant to 2^0 for the least significant.

In representing integer values, the minimum number of octets possible must be used. The most significant nine bits of the contents must not all be the same (either all 1s or all 0s), as such a value could be represented by (at least) one fewer octet.

Thus, given the integer value of -27066 to send, the value is encoded thus:

02029646_{16}

and can not be encoded as, for example

$0204FFFF9646_{16}$

Appendix D Schematic Diagram Generated by SYNOPSIS synthesis tool (See 6.1.3)

Appendix E Timeline Snapshot (See 7.3.3)

Appendix F Simulation Results for POBA (See 7.4)

Figure 49 present throughput performance of POBA with 10 processors under varying workloads — exponential and uniform distribution, both with a mean of 2048 octets. The results obtained are approximately equal to those for a fixed length workload of 2048 octets. Each point in the graph represents the mean value obtained for a series of five runs (A confidence level of 95% is achieved with the maximum calculated confidence interval of ± 5.4 Mbps). This demonstrates that POBA is well balanced even under large workload variations. Figure 50 shows throughput results for POBA with 1, 3 and 5 simultaneous connections. The total throughput of the system remains relatively stable even when the number of active connections is increased. Figure 51 shows the effect of switch in processing path for POBA with 8 processors. Figures 52 to 60 show throughput, control processor utilization and average PPN utilization results for POBA with 4 Kbyte packets.

Figure 49 POBA under various Workloads

POBA with 10 processors and $P = 1024$ octets

Figure 50 POBA with multiple active Connections
POBA with Message length = 1Kbytes; $N = 8$; $X = 4$

Figure 51 Effect of switch in processing path for POBA with $N = 8$

Message Length = 4Kbytes

Figure 52 3D plot of Throughput versus X and N (4 Kbytes)
Message Length = 4Kbytes

Figure 53 Throughput versus X (4 Kbytes)

Figure 54 Throughput versus X (4 Kbytes)

Message Length = 4Kbytes

Figure 55 3D plot of Control Processor Utilization versus X and N (4 Kbytes)

Message Length = 4Kbytes

Figure 56 Control Processor Utilization versus X (4 Kbytes)

Figure 57 Control Processor Utilization versus N (4 Kbytes)

Message Length = 4Kbytes

Figure 58 3D plot of PPN Utilization versus X and Y (4 Kbytes)
Message Length = 4Kbytes

Figure 59 PPN Utilization versus X (4 Kbytes)

Figure 60 PPN Utilization versus N (4 Kbytes)

END

0 8-1 1-94

FIN