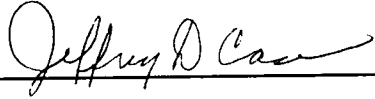


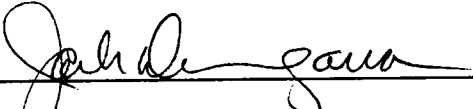
To the Graduate Council:

I am submitting herewith a dissertation written by Subhendu Ghatak entitled "Evaluation Methods for Internet Management Schemes." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

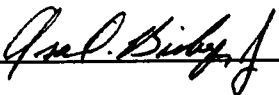


Dr. Jeffrey D. Case, Major Professor

We have read this dissertation
and recommend its acceptance:







Accepted for the Council:



Associate Vice Chancellor
and Dean of The Graduate School

Evaluation Methods for Internet Management Schemes

A Dissertation
Presented for the
Doctor of Philosophy Degree
The University of Tennessee, Knoxville

Subhendu Ghatak
December 1992

Acknowledgements

I thank my advisor Dr. Jeffrey D. Case for his guidance and support. I also thank the members of my committee, Dr. Asa O. Bishop Jr, Dr. Jack Dongarra, and Dr. Tom Dunigan for their interest and advice.

It is impossible to name all the individuals who have helped me during the completion of this dissertation. I thank Ken Key for patiently answering me each time a question cropped up. I thank the wizards of the computer science laboratory for their continuous support and friendship. Dr. Jiliang Wang helped write the MIB parser used in this project. Mike Davison helped me to modify a device driver. Their assistance is greatly appreciated.

I thank Dr. Shashikant Mulay, Debaprasad Kastha, and Vijay Krishna for all their help. I thank the members of the Department of Computer Science for providing a supportive and stimulating environment during the completion of this work. I would also like to acknowledge the services provided by the University of Tennessee's John C. Hodges library.

I thank the Department of Computer Science, Martin Marietta Corporation, and Digital Equipment Corporation for providing me with financial support.

Finally, I thank my parents Taraprasanna and Mira Ghatak, and my grandmother, Kamala Ghatak, for their constant encouragement.

Abstract

This research seeks to identify a systematic procedure to evaluate network management frameworks. The methods developed here are based on the need to make a comparative evaluation of the ISO and Internet network management frameworks, as specified by the Common Management Information Protocol (CMIP) and the Simple Network Management Protocol (SNMP) respectively. The basic principle underlying the synthesis of these evaluation procedures is the separation of the services provided by the management framework into groups that are likely to exhibit different characteristics with reference to the evaluation measures. It is expected that these procedures can also be applied to other management frameworks that employ a query response model. However, the relative evaluation of two management protocols is only possible to the extent they support common sets of management operations.

The evaluation of the physical, data link, network, and transport protocols has received considerable attention in the literature. In contrast, the performance analysis of the upper layers (session, presentation, and application) has remained virtually untouched. Since network management protocols reside in the application layer, this work represents one of the first attempts to evaluate an upper layer protocol.

CMIP has been implemented over the ISO Development Environment (ISODE) platform as a part of this project. The evaluation process covers the agent's performance and host impact, as well as the management framework's response in lossy environments, and its impact on the network. Stochastic models have been introduced to track the behavior of the connection oriented and connectionless network management schemes in lossy environments.

Application of the evaluation methods to CMIP and SNMP have led to a number of interesting results. It is shown that scoping in CMIP can be CPU intensive unless a proper scoping method is chosen. Using the Remote Operations Service of ISO requires a large amount of code, especially since the RO specification of CMIP is complex.

The “semantic gap” between the physical encoding of data and that which can be conveniently used by the management agent at the application layer leads to a large amount of code for translating back and forth between the two representations. It is shown that the connectionless approach is necessary for network management when packets are dropped. It is also found that the linked reply facility of CMIP minimizes network impact when the network is not lossy. Recommendations are provided for the construction of future network management protocols.

Contents

1	Introduction to the Problem	1
1.1	Introduction	1
1.2	Statement of the Problem	2
1.3	Significance of the Study	2
1.4	The Research Questions	3
1.5	Applicability and Limitations of the Study	3
2	Background and Related Research	5
2.1	The Internet Management Protocols	5
2.1.1	The Host Management Protocol	5
2.1.2	The High-Level Entity Management Protocol	6
2.1.3	The Simple Network Management Protocol	7
2.2	The CMIP Based Network Management Architectures	8
2.2.1	CMIP Over TCP/IP (CMOT)	8
2.2.2	CMIS over ISODE	9
2.3	System Evaluation Efforts in Related Areas	10
2.4	Modeling Computer Communication Networks	12
3	Research Procedure	14
3.1	Question 1: Evaluating the Management Framework's Behavior in a Lossy Environment	14
3.2	Question 2a: Evaluating the Performance of the Agent	15
3.3	Question 2b: Evaluating the Memory Requirements of the Agent	17
3.4	Question 3: Evaluating the Management Framework's Impact on the Network	18
3.5	Coding Requirements	18
4	The Management Framework's Behavior in a Lossy Environment	19

4.1	General Introduction	19
4.2	Objective and Methodology	21
4.2.1	Modeling Management Transactions over a Lossy Network	22
4.2.2	Solving a Markov Chain Model: an Example	24
4.2.3	The Experimental Environment	26
4.3	Evaluating a Connection Oriented Management Protocol	26
4.3.1	The Association Establishment Phase	27
	Description of the Model	27
	Assigning Probabilities to the State Transitions	32
	Validating the Model	32
	Association Establishment Under High Loss	36
4.3.2	The Data Transfer Phase (One-Query-One-Object)	39
	Validating the Model	41
	Data Transfer (One-Query-One-Object) Under High Loss	42
4.3.3	On Separating the Association Establishment and the Data Transfer Phases	45
4.3.4	A Simple Approximation of the Protocol's Packet Requirements	47
4.3.5	The Data Transfer Phase (One-Query-Multiple-Objects)	50
	Introduction to CMIP's Linked Replies	50
	Data and Analysis	51
	Conclusions	54
4.4	Analysis of the Connectionless Transport Protocol	56
4.4.1	A Model of the Connectionless Mode of Message Exchange	56
	Experimental Validation of the Model	59
	Conclusions	60
5	Evaluation of Agent Performance	62
5.1	Objectives and Methodology	62
5.1.1	Determining the Sample Size	64
5.1.2	Selecting the Test Object Classes	66
5.1.3	Formulating the Basic Operations	68
	Formulating Tests for the GET Operation	68
	Formulating Tests for the SET Operation	70
	Formulating Tests for the CREATE and DELETE Operations . .	71
	Formulating Tests for the EVENT-REPORT Operation	71
5.1.4	Controlling the Experimental Environment	71

	Factors External to the Manager and Agent	72
	Factors Internal to the Manager and Agent	75
5.2	The Performance Data and its Analysis	76
5.2.1	Retrieval Time for a Single Instance of an Object Class	77
5.2.2	Retrieval Time for an Instance Each from Two Different Object Classes with Scoping and Filtering	77
5.2.3	Retrieval Time for a Single Attribute	81
5.2.4	Retrieval Time for an Attribute Each from Two Different Object Classes with Scoping and Filtering	83
5.2.5	Retrieval Time for the Entire Column of a Tabular Object	85
5.2.6	Retrieval Time for Entire Tables	87
5.2.7	Time to Set a Single Attribute	89
5.2.8	Time to Set a Row Entry	89
5.2.9	Time to Create an Instance	92
5.2.10	Time to Delete an Instance	92
5.2.11	Time to Generate a Trap	95
5.3	Research Question 2c: Aspects of CMIP and SNMP Implementations that Lead to Poor Performance	95
5.4	Conclusions	99
6	Evaluation of the Agent's Memory Requirement	101
6.1	Objectives	101
6.2	Controlling the Experimental Environment	101
6.3	The Text Segment	102
6.3.1	Measurement Strategy	102
	The Components of a Network Management Agent	103
6.3.2	OSI Remote Operations	105
6.3.3	Data and Analysis	106
	The Text Segment of the CMIP Agent	106
	The Text Segment of the SNMP Agent	108
6.3.4	Closing Remarks	110
6.4	The Heap Segment	110
6.4.1	The Workload	110
6.4.2	Measurement Strategy	111
6.4.3	Data and Analysis	116
	The Heap Segment of the CMIP Agent	116

The Heap Segment of the SNMP Agent	124
6.4.4 Closing Remarks	130
6.5 The Data Segment	131
6.5.1 Measurement Strategy	131
6.5.2 Data and Analysis	132
The Data Segment of the CMIP agent	132
The Data Segment of the SNMP agent	133
6.6 The Stack Segment	134
6.6.1 Measurement Strategy	134
6.6.2 Data and Analysis	134
6.7 Research Question 2d: On CMIS's Functional Unit Groups	135
6.8 Conclusion	139
7 The Management System's Impact on the Network	140
7.1 Objectives and Methodology	140
7.2 The Number of Packets Generated by the System	141
7.2.1 Association Establishment	141
7.2.2 One-Query-One-Object Interactions	142
7.2.3 One-Query-Multiple-Objects Interactions	142
7.2.4 Association Release	144
7.3 The Number of Bytes Generated by the System	145
7.3.1 One-Query-One-Object Interactions	145
Findings	146
7.3.2 One-Query-Multiple-Objects Interactions	147
Findings	147
7.4 Conclusions	149
8 Summary, Conclusions, and Recommendations	150
8.1 Summary and Conclusions	150
8.1.1 Question 1: Evaluation of the Management Framework's Behav- ior in a Lossy Environment	150
Evaluation Strategy	151
Conclusions	151
8.1.2 Question 2a: Performance Evaluation of the Agent	152
Evaluation Strategy	152
Conclusions	152

8.1.3	Question 2b: Evaluation of the Agent's Memory Requirement . .	153
	Evaluation Strategy	153
	Conclusions	154
8.1.4	Question 2c: What are the host impacts of the different func-	
	tional unit groups of CMIP ?	155
8.1.5	Question 2d: What are the bottlenecks in CMIP and SNMP	
	implementations?	156
8.1.6	Question 3: Evaluation of the Management Framework's Impact	
	on the Network	156
	Evaluation Strategy	156
	Conclusions	157
8.2	Recommendations	157
8.2.1	Recommendations for Users of Network Management Protocols .	158
8.2.2	Recommendations for Designers of Network Management Protocols	158
8.2.3	Recommendations for Implementors of Network Management	
	Protocols	159
8.2.4	Recommendations for Future Study	160

List of Figures

2.1	The Internet network management protocols.	6
2.2	The CMOT architecture.	8
2.3	The relation between system parameters (metrics) and basic workload components.	11
4.1	An example Markov chain.	24
4.2	The association establishment phase of CMIP — a connection-oriented management protocol.	28
4.3	A Markov model for the association establishment phase.	29
4.4	A model for one-query-one-object transactions.	40
4.5	Connection-oriented transaction over a lossy net (left) and a lossless net (right).	48
4.6	The percentage of timeouts vs. the packet drop rate.	55
4.7	The connectionless mode of data transfer.	57
4.8	Observed activity completion times.	60
5.1	The GET operation formats used in performance evaluation.	69
5.2	CMIP and SNMP data for basic operation 1.	78
5.3	CMIP and SNMP data for basic operation 2.	80
5.4	CMIP and SNMP data for basic operation 3.	82
5.5	CMIP and SNMP data for basic operation 4.	84
5.6	CMIP and SNMP data for basic operation 5.	86
5.7	CMIP and SNMP data for basic operation 6.	88
5.8	CMIP and SNMP data for basic operation 7.	90
5.9	CMIP and SNMP data for basic operation 8.	91
5.10	CMIP and SNMP data for basic operation 9.	93
5.11	CMIP and SNMP data for basic operation 10.	94
5.12	CMIP and SNMP data for basic operation 11.	96

5.13	Filtering done on the agent (top) and on the manager (bottom).	100
6.1	Sampling occurring at times when the heap size is sub-maximum.	111
6.2	The effect of incidental allocations on the maximum heap size.	113
6.3	Increase in the MHS_{mod} values during the execution of a CMIP agent.	122
6.4	Increase in the MHS_{mod} values during the execution of an SNMP agent.	129
6.5	A situation where sampling occurs when the stack size is sub-maximum.	134
7.1	The number of objects retrieved vs. the total number of packets required.	143
7.2	The number of bytes required vs. the number of attributes retrieved by a query.	146
7.3	The number of bytes required vs. the length of the IP routing table.	148

List of Tables

1.1	The agent types as defined in RFC 1095.	4
4.1	Observed and theoretical data for the association establishment phase. .	34
4.2	The number of packets and the time required to complete association establishment, $q = 0.6$	37
4.3	The number of packets and the time required to complete association establishment, $q = 0.7$	37
4.4	The number of packets and the time required to complete association establishment, $q = 0.8$	38
4.5	P_A and T_A values for various values of $q > 0.5$	38
4.6	Theoretical and experimental data for one-query-one-object transactions.	42
4.7	Observed data for one-query-one-object transactions, $q = 0.6$	44
4.8	Observed data for one-query-one-object transactions, $q = 0.7$	44
4.9	Observed data for one-query-one-object transactions, $q = 0.8$	44
4.10	The P_{total} and T_{total} values for $q = 0.6$ to 0.8	45
4.11	Comparing the approximation $\frac{m}{p}$ with observed data.	47
4.12	Observed data for one-query-one-object transactions.	51
4.13	Observed data for one-query-multiple-object transactions, $q = 0.6$	52
4.14	Observed data for one-query-multiple-object transactions, $q = 0.7$	52
4.15	Observed data for one-query-multiple-object transactions, $q = 0.8$	52
4.16	The P_{total} and T_{total} values for $q = 0.6$ to 0.8	54
4.17	The mean number of packets (including association establishment and data transfer) required by the two modes of data transfer.	55
4.18	The number of packets and the time required to retrieve 10 objects with SNMP.	59
5.1	Confidence intervals of the sample means at a 95% confidence level. . .	66

5.2	The number of instances for the tabular objects.	74
6.1	The number of instances for the tabular objects.	102
6.2	CMIP agent text size.	107
6.3	SNMP agent text size.	109
6.4	CMIP agent heap sizes.	117
6.4	(continued).	118
6.5	CMIP heap requirements (Kbytes) at idle state.	120
6.6	SNMP agent heap sizes.	125
6.6	(continued).	126
6.7	SNMP heap requirements (Kbytes) at idle state.	128
6.8	The CMIP data segment.	132
6.9	The SNMP data segment.	133
6.10	Additional text required to move from a simple to a controlled agent. .	136
6.11	Additional text required to move from a monitored to a simple agent. .	137
6.12	Additional text required to upgrade from an event sender to a monitored agent.	137
7.1	The number of packets and bytes required by the association establish- ment phase of CMIP.	141
7.2	The average number of packets required vs. the number of objects re- covered using a OQMO type of query.	143
7.3	The number of packets required to retrieve a managed object.	144
7.4	The number of packets and bytes required by the association release phase of CMIP.	145

Chapter 1

Introduction to the Problem

1.1 Introduction

Computer networks fulfill a variety of vital functions in various fields. It is therefore important to manage these networks in order to ensure their availability and satisfactory performance. The Internet community fielded the Simple Network Management Protocol (SNMP) [16] as an Internet management standard in 1988. In the ISO-OSI (International Organization for Standardization – Open Systems Interconnection) model, the Common Management Information Services (CMIS) [1] and the Common Management Information Protocol (CMIP) define the necessary internet management facilities. Though CMIP and SNMP provide the same basic services, CMIP does have some additional features and options. It is expected that this added versatility will incur high overheads in terms of the code length and time necessary for each management operation. This overhead is unacceptable for resource feeble systems like bridges Ethernet hubs. CMIP's use of a connection-oriented (CO) transport protocol also makes it difficult to reach a remote host if the packets are lost. In such situations, effective management requires a quick approach. SNMP has already proved effective in this regard. If CMIS/CMIP is to become widely acceptable, it has to be capable of tackling such situations speedily and with minimum overhead. This leads to the general problem of making an objective comparison of various network management protocols. Much has been said about the relative merits and demerits of the various approaches to network management. However, there have been very few reported attempts to *systematically* evaluate the performance of network management protocols. This work uses the terminology of the OSI network management framework. Familiarity with the CMIS/CMIP standards will therefore be helpful.

1.2 Statement of the Problem

This research identifies and utilizes methods to evaluate the agent's performance, its storage requirements, the management framework's behavior in a lossy environment, and its impact on the network. This work will therefore define systematic evaluation methods for network management protocols which use a management information base (MIB) and a request-response style of interaction to effect management.

1.3 Significance of the Study

There has been a proliferation of network management services and protocols in recent years. However, there are no methods for systematically determining the efficacy and performance of these protocols. As a case in point, a recent document [31] observes that the Netlabs¹ CMIP over TCP/IP (CMOT) agent has been benchmarked at 100 management operations per second (MOPS). Their SNMP agent however performed at 200 MOPS. The tests were made on a machine capable of performing a million instructions per second (MIPS). The exact nature of the evaluation process was not reported. The basic principle underlying the synthesis of the evaluation procedures developed here is the separation of the services provided by the management framework into groups that are likely to exhibit different characteristics with reference to the evaluation methods. Such methods may then be applied to all management frameworks that follow a request-response model.

The advantages of having such well defined evaluation methods are immediately apparent. In broad terms, they are as follows:

1. The designers and implementors of network management protocols will have a set of criteria to fulfill. This will lead to better protocol designs and implementations.
2. The user can objectively compare different approaches to the network management problem.

Another important issue is that the performance of the lower layers, i.e., layers one through four, of the conventional protocol stacks have been studied in detail [58]. However, the upper layers, i.e., the session, presentation, and application layers, have not received much attention because they are not readily amenable to analysis by the established network modeling tools such as queuing networks [49], Markov chains [61, 62], or

¹Netlabs is a trademark of Netlabs Inc., Los Angeles, California.

Petri nets [50, 51]. Liu [54] is of the opinion that for analyzing protocols at the application layer, it is important to categorize the distributed processing tasks performed by the protocol. Since network management protocols are examples of application layer protocols, such an approach has been utilized in the current work. Therefore, this study will be an advance in upper layer protocol evaluation.

1.4 The Research Questions

The research questions will specifically target the CMIP and SNMP network management frameworks. The applicability of this approach to more general situations is described in Section 1.5. The questions are as follows.

1. How does packet loss impact the behavior of the two target network management frameworks?
2. Performance and memory requirements:
 - (a) How do the two target network management frameworks perform?
 - (b) What are the memory requirements of the target network management frameworks?
 - (c) What aspects of CMIP and SNMP implementations lead to poor performance?
 - (d) What are the host impacts of the different functional groups of CMIP?
3. What is the impact of the network management framework on the network?

It may be noted that CMIS can be divided into five functional unit groups as shown in Table 1.1. The answer to question 2a will help determine which agent is most appropriate for any particular system that is to be managed. It is expected that it will be infeasible to construct and use sophisticated CMIP agents for the monitoring and control of many systems. The event sender, simple, monitored, and controlled agents will be considered here. Since standard actions have not yet been defined, the full agent cannot be analyzed at this time.

1.5 Applicability and Limitations of the Study

The methods developed here are based on the need to make a comparative evaluation of the ISO and Internet network management frameworks, as specified by the network

management protocols (CMIP and SNMP respectively) and the associated MIBs and SMIs. As such, they will be directly applicable to network management protocols that employ a MIB based query-response model and support the same set of services provided by CMIP and SNMP.

Table 1.1: The agent types as defined in RFC 1095.

Agent Type	Event Report	Get	Set	Create/Delete	Action	Multiple Reply	Multiple Obj Select
Event Sender	U	no	no	no	no	no	no
Monitored	U	yes	no	no	no	no	no
Simple	U	yes	C	no	no	yes	no*
Controlled	U	yes	U/C	yes	no	yes	yes
Full	U/C	yes	U/C	yes	U/C	yes	yes

C = confirmed, U = unconfirmed

*The simple agent must support a single level of scoping for only those cases where a whole table is to be accessed. The simple agent does not support filters.

However, the principle of separating the services provided by the entity under evaluation into well defined groups is more universal. It is applicable not only to network management schemes but also to other protocols whose service aspects outweigh the protocol aspects (i.e., the message exchange aspect). Therefore, although the basic principle behind the construction of evaluation methods remains identical for a broad range of protocols, the particulars will change as the features of the protocols to be evaluated become more and more distant from the CMIP and SNMP network management schemes which form the basis of the proposed methods.

Chapter 2

Background and Related Research

The need to institute systematic open network management practices that can cover a heterogeneous collection of systems has been felt ever since data communication expanded to multiple networks (internets) that span vast geographical distances. The ability to monitor and control end systems and intermediate systems remotely becomes necessary in such cases. The Internet [68] is a prime example of such a network. Remote management gives one the ability to gain a complete view of what's happening on the net. This facilitates effective control as there is less chance of a local change creating unwanted behavior somewhere else, unknown to the local manager. Although it is the network management application that forms the final interface with the human network controller [5-9], the underlying management services and protocols are of prime importance in determining the effectiveness of any management scheme. The following subsections present an overview of network management protocols that will have a bearing on this project. Key [55] provides a more detailed description of these protocols. Figure 2.1 illustrates the Internet network management protocols.

2.1 The Internet Management Protocols

2.1.1 The Host Management Protocol

In the early days, Bolt Beranek and Newman, Inc. (BBN) was in charge of monitoring and controlling the Internet (then called the ARPANET). The first Internet management protocol was HMP (Host Management Protocol) [10], formulated in 1983 at BBN. This was in response to BBN's need to monitor the PSNs (Packet Switched Nodes) that

made up the ARPANET backbone. HMP is a transport layer protocol that relies on prior knowledge of the nature of the managed entity to carry out its functions. This was due to the absence of any standards for structuring the management information and the lack of a management information base (MIB). The protocol required change every time a new managed object was added to the list of recoverable objects. Thus, in a sense, the HMP was not a standard at all, though the basic concepts laid out therein are similar to those found in present day network management schemes.

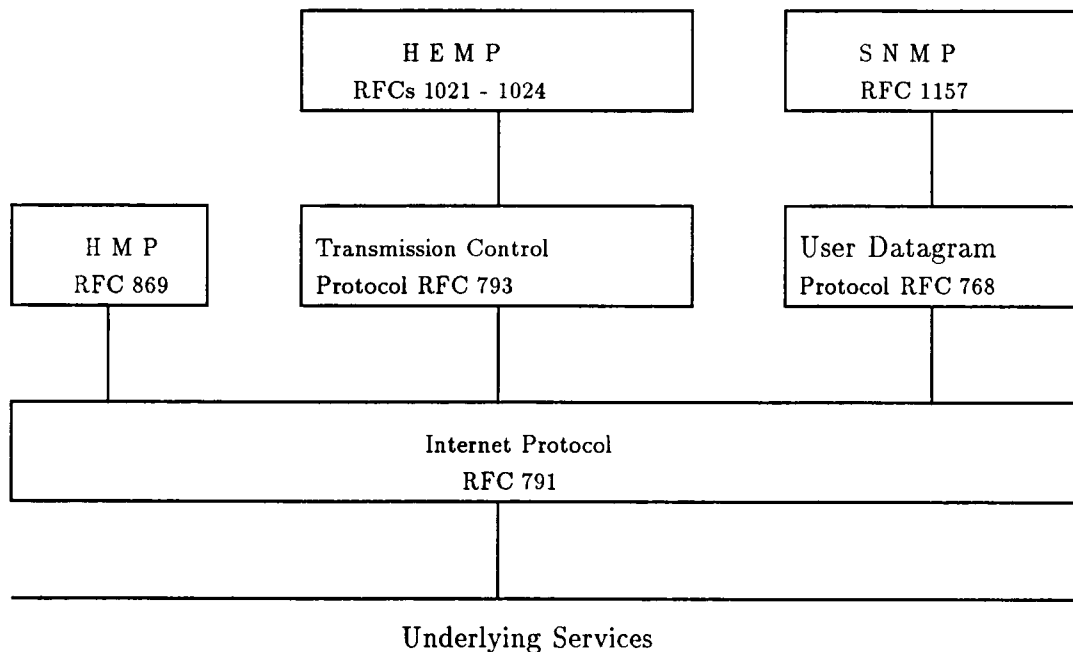


Figure 2.1: The Internet network management protocols.

2.1.2 The High-Level Entity Management Protocol

HMP could not successfully address the requirements of management in environments with machines of diverse nature. Keeping track of the different managed entities (agents) and formulating corresponding HMP packets became difficult. The High-Level Entity Management Service/Protocol (HEMS/HEMP) [11–14] was a noteworthy step towards more standardized and sophisticated management practices. HEMP borrowed certain practices from ongoing OSI network management work and defined a universal

tree-structured type-space, which in OSI or later Internet parlance is known as a MIB. HEMP uses the Abstract Syntax Notation One (ASN.1) data representation rules. No restrictions are placed on the type of management data or on the length of encodings. Key to the HEMS management system is a query processor which interprets all monitoring and control requests. The use of such a query processor resulted from a conscious decision to enable each query to carry multiple instructions. It was felt that this would improve performance. As a result, HEMS responses may span several IP packets. This makes it necessary for HEMP to use reliable stream delivery (as offered by TCP) [11] when the underlying network becomes lossy and it becomes difficult for a packet to reach its destination. Managing a network in such situations is known as network "fire fighting". For routine checks, HEMP may operate over the User Datagram Protocol (UDP). However, HEMP's generality also proved to be its greatest drawback, and it was never implemented widely or deployed.

2.1.3 The Simple Network Management Protocol

Later protocols like the Simple Gateway Monitoring Protocol (SGMP) and the Simple Network Management Protocol (SNMP) are based on the philosophy that network fire fighting is best done with a connectionless (CL) network service [15]. In 1988, SNMP emerged as a quick and effective, albeit relatively simple, way to manage the Internet. At this time, the HEMS/HEMP proposal was withdrawn in order to enable the Internet Architecture Board (IAB) to approve SNMP as an Internet standard without competition and thereby speed up the standardization process [4]. One strong point of SNMP is that all directives and responses are housed in a single datagram [16]. The use of a connectionless transport protocol (UDP) [73] is thereby facilitated. SNMP also uses only the non-aggregate managed objects defined in the Internet-standard MIB [17]. In addition, *all encodings use the definite length form, and non constructor encodings are used as far as possible*. An important feature of SNMP is that only two different protocol data units (PDUs) are required to perform the management operations. These characteristics serve to lower the encoding/decoding overhead that the use of ASN.1 introduces.

2.2 The CMIP Based Network Management Architectures

2.2.1 CMIP Over TCP/IP (CMOT)

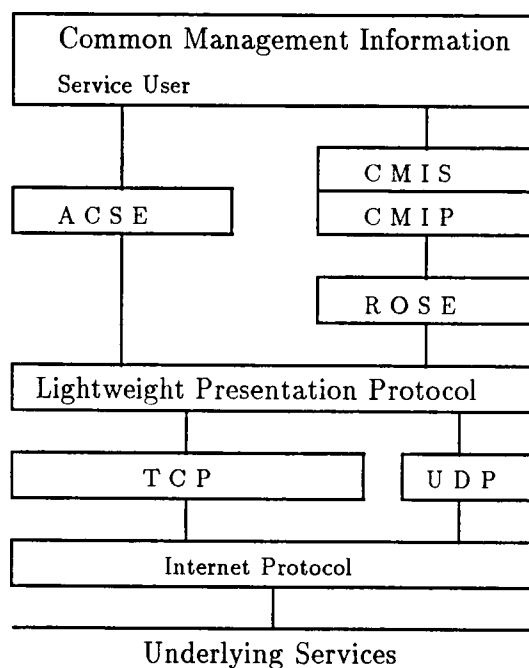


Figure 2.2: The CMOT architecture.

SNMP is the current network management standard for TCP/IP networks. However, CMIS/CMIP is the emerging international standard. Hence the IAB had recommended the use of CMIP over TCP/IP (CMOT) as an interim standard that could be used to gain experience with the ISO network management framework in the Internet environment. The CMOT protocol [18–19] seeks to provide CMIS services over minimal presentation services as provided by the Lightweight Presentation Protocol (LPP) [20]. LPP works over TCP or UDP. The CMOT architecture is shown in Figure 2.2. CMOT also provides a way to interpret the Internet MIB in a manner similar to that dictated by the ISO SMI [21]. This is achieved by imposing additional structure on the managed objects defined by the Internet MIB [17]. To understand this further, it should be noted that the ISO SMI requires object classes to contain the following information:

1. The attributes of the object class,
2. Functions that may be performed on these attributes,
3. Events the object can generate, and
4. The relationships the object is participating in. These relationships are as follows:
 - (a) Containment relationship: A managed object instance of one class may contain other object instances of the same or different classes. The composite, containing object is called the **superior** managed object instance of its **subordinate** managed object instances.
 - (b) Inheritance relationship: One object class may be made an extension of another class by assigning to it additional properties. It also inherits the properties of its parent class. This process is known as **refinement**. The object class thus formed is called a **subclass** of its parent (its **superclass**).

The Internet SMI [22] specifies that managed objects are to be organized as separate items and not as object classes. The CMOT specification retains much of this, but adds a wrapper around it in order to reorganize the managed objects in terms of object classes that closely resemble the object class structure specified in the ISO SMI.

2.2.2 CMIS over ISODE

The desire to gain experience with OSI applications led to the development of the ISO Development Environment (ISODE) [3]. ISODE is a publicly available implementation of the ISO application, presentation, session, and transport (TP0) protocols. TP0 interfaces with TCP through the mechanism described in RFC 1006 [23]. As a part of this project, CMIS/CMIP was implemented using the ISODE platform. The ISODE package (version 6.0) provides tools to encode and decode ASN.1 structures. These routines, though handy, are not efficient. Efforts are continuing to modify the encoding and decoding techniques to achieve greater speed and flexibility [24, 64].

Address translation may be seen as a hindrance to the co-existence of OSI management services over TCP/IP with those over a full OSI stack. In the initial phase of the transition to OSI management methods, CMIS over ISODE systems will communicate with systems of the same nature. Address mapping is not an issue here as IP addresses are used for all communications. However, as full OSI stacks become widely available, there will be a need for translation between the Internet and the OSI addressing styles.

As pointed out in [25], the OSI directory services may be utilized to assist with these mappings.

2.3 System Evaluation Efforts in Related Areas

The need to evaluate network management protocols has been felt for some years now. Ben-Artzi, *et al.* [40] have reported preliminary performance results for CMIP over TCP/IP (CMOT) and SNMP. They also propose a machine independent metric — MOPS/MIPS — for this purpose. The authors state that this single metric is not always indicative of a protocol's performance. The current work provides a solution to this difficulty. A qualitative comparison of, and the implementation requirements for CMOT and SNMP, have also been provided in [40].

A more detailed set of criteria for the evaluation of network management protocols has been presented in Sanghi, *et al.*[42]. This is a list of qualities that are desirable in a network management protocol, and a qualitative discussion of how well SNMP and CMOT fulfill these qualities. Much of the discussion applies to complete network management systems rather than to the network management protocols themselves. A similar approach has been taken by Carter and Dia in [41]. The criteria provided in [41] and [42] provide excellent guidance to the network manager but is of less use to the designer of network management protocols.

Artificial workloads have often been used to evaluate system performance in general [35, 36]. The current work utilizes this evaluation technique. The composition of such workloads depends on the purpose for which it is to be used. As a matter of fact, a set of scripts that direct the evaluator to perform certain specific tasks while conducting the test, may also be included. This is necessary because a performance evaluation experiment must be performed in a controlled environment if it is to yield correct conclusions about the system under test.

It necessary to *characterize* the workload before constructing the tests. This determines which aspects of a workload actually influence the system's performance. These aspects are then termed the workload's *basic components*. In this case, the *system parameters* are variables such as the run time for a management operation and the host impact of a manager or agent (in suitable units) that are affected by varying the basic components. In this situation the selected system parameters will also be the evaluation metrics. The relation between the two has been shown in Figure 2.3. The metrics are identified first, they are then used to identify the basic workload components.

In a performance evaluation study, it is important to know if a test workload ade-

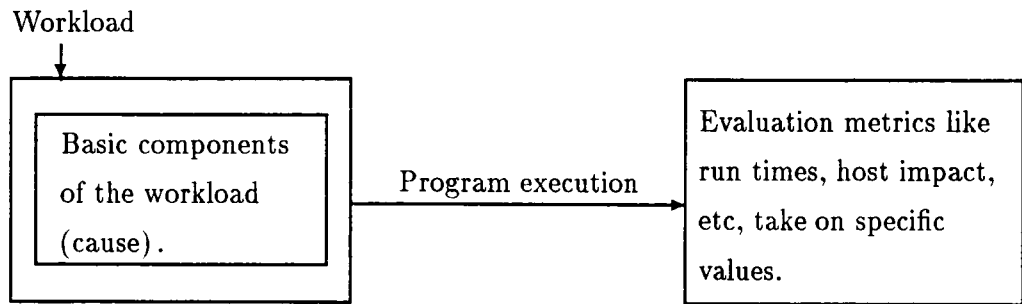


Figure 2.3: The relation between system parameters (metrics) and basic workload components.

quately represents the real workload. The artificial workload is perfectly representative of the real workload if it performs the same functions in the same proportions as the real workload [35]. Although this is a simple criterion, its use is limited, because in many situations the system is driven to an operational extreme rather than to its usual levels. There is therefore no absolute criteria for building and validating workload models.

The following steps are needed to construct artificial workloads. They have been adapted from [35].

1. Definition of the intended use of the model.
 - (a) Exactly what is to be measured?
2. Selection of performance metrics.
3. Workload characterization.
 - (a) Determination of the system parameters to be used.
 - (b) Determination of the basic components and the problem classes that comprise the workload.
4. Selection of a method for representativeness evaluation.
5. Specification of the test environment.
6. Application of these parameter values and the basic components to build a synthetic mix. This is the test workload.
7. Model execution.

8. Application of the adequacy criteria.

- (a) One may proceed with the experiments if the adequacy criteria are fulfilled to a satisfactory extent.
- (b) If the workload is not representative or inadequate in any other way, it is to be modified, and the inadequacy removed, before returning to step 7 above.

The preceding issues guided the selection of the management operations that were used to conduct the evaluations. However, the special nature of the current problem required certain deviations from the norms presented here.

2.4 Modeling Computer Communication Networks

A considerable amount of work has been done in the field of modeling communication networks. Queuing theory and Petri nets are the two principal tools that have found use in network modeling. In his seminal work on the application of queuing networks to computer communications, Kleinrock [49] extensively analyzes message delay and throughput in packet switched networks under various conditions.

Garg [50] has used timed Petri nets to estimate the performance of protocols in terms of the time taken to complete certain operations of the protocol. In particular, she examines the connection and disconnection phases of the European Computer Manufacturer's Association (ECMA) transport protocol. Zubarek [51] provides modifications to timed Petri nets, making them suitable for modeling timeouts in communication protocols.

However, none of these models account for variable time out periods and the closure of a connection when the amount of retransmission crosses specified thresholds.

Reiser [58] provides an overview of the performance evaluation models for the data link, network, and transport layers that are available in the literature. It is important to note that all the models that address retransmission systems assume a constant retransmission timeout in order to make the analysis tractable. Unfortunately, this assumption reduces the applicability of the analysis to protocols like TCP when the underlying network begins to lose packets excessively. In such cases, retransmission will predominate, and the exponential back off scheme followed by TCP will introduce delays far greater than those predicted on the basis of a constant timeout assumption. In addition, none of the current models take into account the fact that a connection might abort itself if too many packets are lost. These shortcomings make existing modeling techniques inapplicable to one of the problems at hand — that is determining

the number of packets and the time required to complete a management task in a lossy network.

Automatic repeat request (ARQ) protocols have received a great deal of attention in the literature [54, 58, 61–63]. Every analysis assumes a constant retransmission timeout.

Petri nets have been used to model the TCP protocol in [60]. However, the model does not deal with TCP's time related aspects. MacLeod [24] provides experimental results on the performance of the ASN.1 encoding and decoding routines of the ISO Development Environment (ISODE). Huitema and Doghri [64] have defined a "light weight" transfer syntax which is reported to provide 1.32 to 6.15 times faster encoding/decoding than the current ASN.1/BER standard.

The preceding models have been developed for the analysis of specific protocols. However, there has been growing interest in developing performance evaluation methods to determine the throughput and delay behavior of communication architectures, taking into account the multiple layers of protocols that are utilized. Conway [65] gives a detailed overview of the methods currently available for this purpose.

Finally, there have been a few studies on application layer protocols. Meister [59] investigates the throughput that can be achieved while transferring files over the ISO transport protocols TP2 and TP4. Yang, *et al.* [66] have modeled the behavior of file transfer over a satellite wide area network. Both studies employ a constant retransmission timeout. The possibility of losing the connection under heavy packet loss is also ignored.

In [67], Nishida presents a modeling framework for end-to-end performance evaluation in a network management system. However, this framework deals more with the protocol layers underlying the management services rather than the management services themselves. This work does not represent how a process executes management operations.

Chapter 3

Research Procedure

The procedures to be followed to address the research questions stated previously are outlined here. General arguments that provide a basis for these procedures are also provided where necessary.

3.1 Question 1: Evaluating the Management Framework's Behavior in a Lossy Environment

Question 1 deals with evaluating the management framework's behavior in a lossy environment. This covers the quality of system response in the face of network degradation. The total number of packets necessary to complete management operations over a lossy net will be calculated. The extent to which a connection-oriented CMIS/CMIP can function in a lossy network is of particular interest.

Since a network that exhibits various degrees of controlled lossiness can not be available for experimentation, a model of such a net is used. There are two general approaches to modeling computer communication systems, analytic modeling and simulation [46]. Both approaches require a somewhat abstract representation of the system being investigated. In general, analytic models require a higher degree of abstraction than simulation models. Analytic models should be used wherever possible since there will be a direct relationship between the model parameters and the performance measures.

The alternative to analytic solution is simulation. This requires the use of a program that behaves like the target system to a necessary degree. The behavior of the program is then observed. The principal advantage of simulation is its simplicity — an arbitrary level of detail can be simulated. The problems are threefold: construction of the

simulation program, the expense of running the program, and the need to statistically analyze the results. Besides, since a direct relationship between simulation model parameters and the simulation output does not exist, additional runs are to be made with different model parameters.

The approach used is to create a model of a lossy network and then validate the model by simulating the lossy net. A simple setup is used to conduct the simulation experiments. The relevant device driver on the manager station is programmed to drop packets going out of or directed to the manager port at a predetermined rate. The quality of the agent/manager connection can be monitored. This provides insight into the connection-oriented and the connectionless models of network management.

3.2 Question 2a: Evaluating the Performance of the Agent

Question 2a deals with evaluating the performance of the agent. The focus here is on the development of certain measures of significance that will reflect the performance of network management protocols. At this point, it is important to realize that:

a network management protocol is a collection of relatively independent services and associated mechanisms that allow the manager to perform certain management operations on remote data objects.

In order to obtain a broad perspective of the protocol's performance, the evaluation process should cover the performance of each service in the context of the management operation it is meant to accomplish.

The GET, SET, GET-NEXT and other similar facilities provided by a management protocol are the services provided to the manager by the protocol. Scoping, filtering, encoding, decoding, etc., are examples of associated mechanisms. An attempt to get the value of a managed object, or to retrieve all the entries of a tabular object are examples of a management operation, or simply an operation. A set of **basic operations** may be defined as possessing the following properties:

1. It is the smallest and simplest set of operations that will exercise all the services and mechanisms offered by the agent.
2. It will cover all the operations that are commonly performed by the management station.

The rationale for such a definition is as follows. The implementor of a protocol may

be interested in knowing how each feature in the protocol performs, hence all the services and features have to be exercised. On the other hand, the designer of the management framework and its user may require knowledge of how the agent behaves at the management operation level, since ultimately the performance at this level will determine how acceptable the management framework will be. The latter requirement has led to the second part of the definition. This definition is subjective. Consequently, the selection of the basic operations requires experience. If experience is unavailable, a few iterations may be necessary before a satisfactory set of basic operations is obtained.

The services and underlying mechanisms provided by the network management protocol exhibit their characteristics only when exercised by a management operation. The exhibited performance of the service will vary with the operation being performed. As an example, the GET service along with the associated scoping mechanism will display different performance characteristics when used to recover different sets of managed objects. The only invariants in this scenario are the basic management operations themselves. Every management protocol must accomplish these operations regardless of the specific services or mechanisms it provides. These basic operations will therefore form the basis of the performance evaluation strategy.

The problem of evaluating the performance of a management protocol therefore reduces to evaluating its performance when applied separately to predefined basic operations which the protocol is to perform with the services and mechanisms best suited for the particular operation. For example, if retrieving all the elements of a tabular object is taken as a basic operation, then CMIP would typically accomplish it by using the GET service along with the scoping mechanism, whereas SNMP would use the GET-NEXT service.

The evaluator would now be in a position to study the experimental results and determine not only how a protocol performs a basic operation, but also how well suited its services and mechanisms are for that operation. Chapter 5 is devoted to formulating suitable basic operations and the analysis of the performance data.

The MOPS/MIPS metric mentioned in [40] is being dropped altogether. This figure is highly sensitive to the exact form of the management query and the object class involved. Any MOPS/MIPS figure will therefore be misleading unless the context is carefully specified. The point of constructing performance measures is not to produce a single number representing the system's performance. The aim is to understand how performance is affected by the various system components and to create a set of realistic expectations for the user [36]. The single MOPS/MIPS number will not achieve this goal. Instead of averaging all measurements to produce a single MOPS/MIPS figure, it

is best to present the data separately for each basic management operation. This will identify the effects of the various protocol services and mechanisms on the management operations.

3.3 Question 2b: Evaluating the Memory Requirements of the Agent

Question 2b deals with evaluating the memory requirements of the agent. In this study, the host impact of the agent represents the amount of memory and CPU time used by the agent. Only the memory requirements is measured under this category. The number of CPU cycles consumed are not considered here as the data gathered in the performance studies will provide the necessary timing information.

The program size itself is not a true measure of the host impact, as during program execution, additional heap and stack space is required. Early in the study, it appeared that the resident set size (RSS) and the number of page faults that occur would be a suitable for quantifying the agent's memory usage. However, it is now clear that this is not a satisfactory measure because:

- Both the RSS and the number of page faults are very system dependent.
- The RSS value provided by the system call `getrusage()` does not always truly reflect the number of resident process pages.
- A major drawback of this measure is that many agents are implemented on disk-less platforms that do not support virtual memory or any overlay techniques. The concepts of resident set size or page faults are not applicable to such systems. Any metric based on them would therefore be useless in these environments.

Consequently, a different metric is employed, namely, the text, data, heap, and stack sizes.

The text and data sizes are obtained by using the Unix `nm` command with the `-n` option. This lists the entries in the symbol table in order of their addresses, thereby enabling easy computation of the address space occupied by text and data segments occupied by the agent. The use of the symbol table will also allow the calculation of the memory used by different sections of the agent, i.e., the encoding/decoding mechanism, the Remote Operations mechanism, the run time MIB representation, etc.

The maximum heap size is obtained by instrumenting the `malloc()`, `calloc()`, and `free()` functions. The maximum stack size is obtained by instrumenting the kernel

function that allocates stack space. These techniques have been further explained in Chapter 6.

3.4 Question 3: Evaluating the Management Framework's Impact on the Network

This question deals with evaluating the management framework's impact on the network. The following are measured for separate operations:

1. The number of packets that go on the wire. For TCP, this may not always be the same for every experiment as the stream will be divided into packets in different ways. The number of packets will represent the impact at intermediate network nodes such as routers and bridges which operate at the packet level. Since each packet typically consumes a fixed length buffer at these intermediate systems, qualitative inferences may be made about how many buffers will be consumed by management traffic.
2. The number of bytes that go on to the wire. This is a measure of network bandwidth that is consumed by a management operation.

3.5 Coding Requirements

As a part of this project, CMIS/CMIP was implemented using the ISODE platform. This involved the modification of a publicly available copy of CMOT, since the structure of the management information is more complex for CMIP. A *lex* and *yacc* based parser has been constructed to process the OSI MIB specification. The implementation of SNMP that is offered with the ISODE did not have the SET service. Code to support this service was obtained from another source and integrated into the existing agent.

Another major area of coding has been the modification of an Ethernet device driver on the agent station. This enabled the agent to drop any given percentage of the packets directed out, towards the manager, or directed in, towards the agent port. Therefore, only the traffic between the manager and the agent is affected. The applications at the other ports do not lose their packets, hence normal operation of the node's other network related functions can continue. A tool was also built to drop and pass packets in any given sequence. This allowed the management connection to be driven to a known state in the Markov models that are introduced in the next chapter.

Chapter 4

The Management Framework's Behavior in a Lossy Environment

4.1 General Introduction

This chapter studies the ability of a network management protocol to carry out management activities when the underlying network is lossy, and therefore a certain percentage of the management packets are lost in transit. *This is measured by the total expected number of datagrams and the time required to complete a management activity given that each datagram has a certain probability of being dropped.* This will depend on the nature of the management protocol (i.e., CO or CL), the underlying transport protocol, the protocol's own scheme of dealing with lost messages, and the total number of packets that are necessary to complete the management activity in a lossless situation.

In this chapter the following situations will be considered:

1. a connection-oriented management protocol running over a CO transport protocol, and
2. a connectionless management protocol running over a CL transport protocol.

A connection-oriented transport protocol will be discussed first. TCP provides a relevant example because the ISODE stack functions over TCP/IP. In the event of lost packets one of two things may occur:

- Some of the packets must be retransmitted to complete the transaction. The total number of packets required to complete a transaction in a lossy network will

depend on the number of packets required to perform the same function under no loss conditions and the loss rate. This obviously means that a greater number of packets are required for a connection-oriented protocol, because it requires additional acknowledgement packets to complete a transaction. Since these acknowledgement packets may also be lost, it will lead to more retransmissions. The lossy network models described later will clarify these details.

- The transport connection may be reset if one side has to retransmit a segment too many times. Such events are detected and reported to the application, which may then attempt to reestablish the transport connection.

If a connectionless transport protocol is used, there is no question of the connection being reset since no explicit connection exists. The manager may compensate for lost packets by retransmitting queries whose response is not received within a certain time. There should also be some mechanism on both the manager and the agent to deal with duplicate responses and queries respectively. Such measures are usually not part of the management protocol specification itself and therefore depend entirely on the implementation. In any case, since separate acknowledgement packets are not necessary, the total number of packets needed to complete a management transaction is reduced. Hence the total number of retransmissions necessary are also reduced.

The preceding discussion might give one the impression that a connectionless transport protocol, with the manager retransmitting queries as necessary, is better in the sense that the total number of datagrams required to complete a transaction will be less, especially in a lossy network. Although this is true, for a management protocol to make full use of this situation, the queries and responses must also fit into single datagrams. This is because the underlying network operates in terms of datagrams. If one datagram is lost, the corresponding query can be retransmitted by the manager. However, if the response spans multiple packets (the queries themselves are short and almost certain to fit into a single datagram), a lost datagram will force the manager to repeat the entire query and thereby elicit the same multi-datagram response from the agent. The manager has no way of asking for only the information that was lost. Since multiple datagrams are involved, the probability of dropping at least one of them, thereby losing the entire response, remains higher than the probability of losing a single datagram, $(1 - p^n)$ against $(1 - p)$, where p is the probability of a packet reaching its destination successfully and n is the number of datagrams being transmitted.

4.2 Objective and Methodology

The objective of this chapter is to evaluate the management framework's behavior in a lossy environment. This is done by measuring the total number of packets and the time required to complete a management activity over a lossy network.

It may be recalled that the basic principle underlying the synthesis of the evaluation procedure is the separation of the services to be evaluated into groups that are likely to exhibit different characteristics with reference to the evaluation measures (i.e., packet count and time in this case). With this in mind, the services to be evaluated are partitioned as follows.

1. Connection-oriented management protocol (CMIP).

- (a) Association establishment.
- (b) One-query-one-object (OQOO) interactions. A single reply will be returned to each query in a single packet.
- (c) One-query-multiple-object (OQMO) interactions. Multiple objects may be returned in response to a single query. More than one packet may be used to transmit the response.

2. Connectionless management protocol (SNMP)

The connection release phase of a connection-oriented management protocol will not be addressed. This phase is of little concern to the manager for at least two reasons.

- If the manager is able to correct the problem, the packet loss rate will drop to normal levels and connection release will progress normally.
- If the net is still lossy, the connection will eventually be aborted without the manager's intervention.

In order to make a detailed study of the behavior of the management protocols under various levels of packet loss, such behavior can be conveniently modeled as an absorbing Markov chain [45]. The next two subsections provide the groundwork for a Markov chain model. These models are applicable to situations where the rate of packet loss is relatively low. The models predict the total number of packets required to complete an activity. Experimental data is used to evaluate the management protocols under conditions of heavy packet loss. The environment for these experiments is specified in a subsequent section. The time requirements are evaluated experimentally in each case. The connectionless situation can be modeled analytically, as it exhibits less complex behavior.

4.2.1 Modeling Management Transactions over a Lossy Network

In order to better interpret the experimental data, it is important to make a model of the lossy network. A lossy network running either a CL or a CO transport protocol can be represented by a Markov chain under certain simplifying assumptions. The mathematics of Markov chains also provides the tools necessary to compute the expected number of retransmissions necessary in a lossy network.

The following assumptions are necessary for the construction of these models:

1. If a connection-oriented transport protocol is used, then the transport connection will not be reset during the process. Hence the CO models will only be valid for low (up to 50%) rates of packet loss for which this assumption holds good over most trials.
2. Each query and response will be contained in a single packet.
3. No IP fragmentation is necessary. This guarantees that the probability of losing an IP packet is the same as the probability of losing a TCP segment or a UDP datagram. As an example, consider a datagram that is broken into multiple fragments. IP would discard this datagram if at least one of the constituent fragments (packets) is dropped. The probability of dropping at least one out of a number of packets is more than the probability of dropping any one packet, hence the probability of dropping the entire datagram is also higher than that of dropping a single packet under these circumstances. The model avoids this situation in the interest of simplicity. The terms segment, datagram, and packet will therefore be used interchangeably in this section.
4. The probability of being dropped is the same for all packets. This means that the statement, "x% of the packets are dropped" is equivalent to the statement, "the probability of each packet being dropped is x%. This assumption will be used throughout this analysis. At this point one could legitimately wonder if all packets really have the same probability of being dropped or if this probability is related to the size of the packet. The question may be answered as follows:

There are two main reasons for dropping a packet:

- (a) lack of buffer space at the destination host or at some intermediate point,
or
- (b) a checksum error due to the packet being corrupted by line noise.

The probability of a packet being affected by line noise varies with its length. However, since today's technology provides a good signal-to-noise ratio, the occurrence of such errors is rare. The lack of sufficient buffer space is therefore of greater interest. If buffer space is allocated dynamically, depending on the size of the packet, then larger packets are more likely to be dropped. However, it is simpler to use fixed size buffers. Most systems keep a pool of fixed size buffers for this purpose. The probability of a packet being dropped therefore becomes independent of its size, and may be assumed to be approximately equal for all packets.

5. Packets are dropped only due to failures that occur below the transport layer. A packet reaches the application if the transport layer receives it correctly.
6. Packets may be dropped but not delayed. This simplifies the construction of the models because the number of possible states decreases when delayed packets are not considered.
7. If the transport entity (TCP, etc.) on one node has dispatched a packet, then the peer transport entity does not send any packet of its own until the packet in transit has had enough time to reach it. The arcs in the following models also denote the passage of this transit time. The self loops however represent round-trip times, since retransmission occurs only when a transport entity fails to hear from its peer within the round-trip time.
8. An important assumption is that the probability of transition from state i to state j at the n th transition, $p_{i,j}(n)$, is independent of n . This is not strictly true. However, one may use an approximate transition probability which will be assumed to remain constant over time. The results derived with such an approximation will be sufficiently accurate for the purpose at hand.
9. Keepalive packets are not used.

The behavior of a connection-oriented transport protocol will be different during the connection establishment and data transfer stages. Each stage will therefore be modeled separately. The connection release stage is not considered due to reasons described at the beginning of Section 4.2.

The following are common notational conventions used in all the models:

- The probability of dropping a packet is q . The probability of a packet reaching

its destination is $p = 1 - q$. State i may be written as “state s_i ” or just “ s_i .” The probability of moving from s_i to s_j is $p_{i,j}$.

4.2.2 Solving a Markov Chain Model: an Example

This section illustrates the method of obtaining the mean (μ_0) and standard deviation (σ_0) of the number of packets that are required to complete an activity over a lossy net. The complete theoretical background may be found in [45].

Consider the example of a Markov chain shown in Figure 4.1. The process reaches completion, or is absorbed, when it reaches the final state s_4 . This state is also called the *absorbing state* because once the process enters s_4 , it cannot leave. States s_1, s_2 , and s_3 are the *transient states*. The label on each arc represents the probability of transition to one state from another. The sum of the probabilities on all the arcs exiting a state should therefore be 1, i.e., $p + q = 1$.

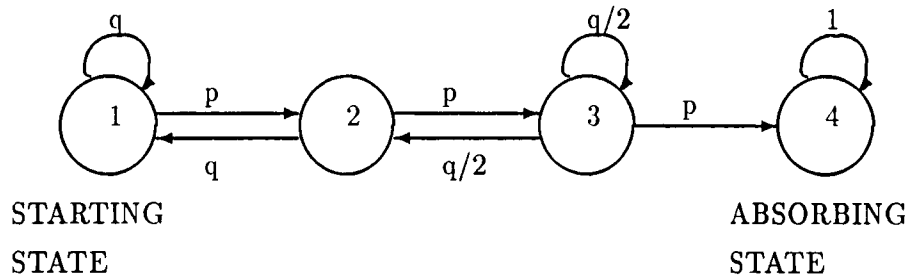


Figure 4.1: An example Markov chain.

Consider the transition matrix for the chain in Figure 4.1.

$$\begin{array}{c}
 \begin{array}{c} s_4 \\ s_1 \\ s_2 \\ s_3 \end{array}
 \left[\begin{array}{c|ccc}
 & s_4 & s_1 & s_2 & s_3 \\
 \hline
 s_4 & 1 & 0 & 0 & 0 \\
 s_1 & 0 & q & p & 0 \\
 s_2 & 0 & q & 0 & p \\
 s_3 & p & 0 & q/2 & q/2
 \end{array} \right]
 \end{array}$$

The part of the transition matrix that corresponds to the transient states is now written as

$$Q = \begin{bmatrix} q & p & 0 \\ q & 0 & p \\ 0 & q/2 & q/2 \end{bmatrix}$$

The fundamental matrix for this system is then defined as

$$N = (I - Q)^{-1}, \quad I \text{ is the identity matrix.}$$

The element N_{ij} of this matrix is the mean total number of times the process is in state s_j before being absorbed, s_i being the starting state. Therefore, when s_1 is designated as the starting state

$$N_{total} = N_{11} + N_{12} + N_{13}$$

will be the mean total number of times the process is in a transient state before being absorbed. If P_i packets are transmitted in the transient state s_i , then μ_0 , the mean number of packets transmitted by the process before completion is given by

$$\mu_0 = \sum_{i=1}^t P_i N_{1i}, \quad t \text{ is the number of transient states.}$$

The following steps are required to compute the variance of each N_{ij} . Let

$$N_2 = N(2N_{dg} - I) - N_{sq}$$

Here N_{dg} is a diagonal matrix such that $(N_{dg})_{ii} = N_{ii}$, and N_{sq} is a square matrix such that $(N_{sq})_{ij} = (N_{ij})^2$. $(N_2)_{ij}$ is the variance of N_{ij} .

In some models, the number of times a process is in a particular transient state may be assumed to be independent of, or have negligible correlation to, the number of times it will be in any other transient state. The variance of N_{total} can then be written as the sum of the variances of N_{11} , N_{12} , and N_{13} , i.e.,

$$var(N_{total}) = (N_2)_{11} + (N_2)_{12} + (N_2)_{13}$$

However, the variance of the total number of transmitted packets will be given by

$$\sigma_0^2 = \sum_{i=1}^t P_i^2 (N_2)_{1i}, \quad \text{where } t \text{ is the number of transient states.}$$

The value of σ_0 follows.

4.2.3 The Experimental Environment

All the experiments reported in this chapter were conducted in the environment specified below.

1. The manager and the agent were running on two separate Sparcstation 2s, located adjacent to each other on the same Ethernet cable. Packet transmission time is therefore negligible.
2. All remote logins were disabled. This measure prevents contamination of the experimental results by unrelated activities on the nodes acting as the agent and the manager. It would still be possible to login remotely as a super user or to use the remote shell facility, but this is rare in this environment.
3. X-Windows was running on both the manager and the agent, with two windows on the agent platform and four on the manager platform.
4. No other jobs were running on the agent node, except the agent software itself.
5. The manager node had a remote login session underway on a third workstation in order to run the *etherfind* utility. This was to watch the flow of packets between the manager and the agent. Although this would cause increased TCP activity on the manager, the impact was hardly perceptible, especially since the timing data has a large granularity of 1 second. No other process was active on the system running the manager.
6. The usual Unix daemons like *inetd*, *lpd*, etc were running. Their presence, however, had no perceptible effect on the measured times, since they did not undergo significant activity during the measurement session.

4.3 Evaluating a Connection Oriented Management Protocol

The evaluation of a connection oriented management protocol is partitioned into the analysis of the association establishment phase, followed by the analysis of the data transfer phase. CMIP, the connection oriented management protocol, supports two primary modes of data transfer: one-query-one-object and one-query-multiple-objects. Each mode is studied under both low and high loss conditions. The effects of the partitioning of the analysis by phase are also considered.

4.3.1 The Association Establishment Phase

The association establishment phase requires all the layers from the transport layer up to the association control layer to set up an association with their peers. This is best seen in an OSI association since all the seven layers exist. In the Internet suite of protocols, only the transport layer (TCP) needs to set up a connection via the three-way handshake. The behavior of both these mechanisms can be observed through study of the ISODE association establishment process, since it is layered over TCP. Hence ISODE, and therefore CMIP, is required to perform a three-way handshake before it proceeds with the rest of the association establishment mechanism. The aim here is to determine the total number of packets and the time necessary to establish an association in a lossy network. The mean number of packets to completion (MNPC) will characterize the management framework's behavior over a lossy net. The models do not account for the possibility of a connection timeout. A timeout occurs when the amount of retransmission of a given segment exceeds a specified threshold. The threshold may be measured in time units or as a count of retransmissions. Hence, to calculate the chances of a timeout, one should calculate the mean number of retransmissions of each segment and relate it to the threshold value. The creation and validation of an extended model is deferred to future research.

Description of the Model

Figure 4.2 illustrates the normal operation of the association establishment phase under zero packet loss. This is the basis of the Markov chain model of Figure 4.3, which simplifies to the situation shown in Figure 4.2 when $p = 1$. The model was constructed from prior knowledge of the protocols' behavior and by observing a packet trace of the association establishment phase. It was also necessary to force the net to drop particular packets and observe the subsequent behavior of the protocols.

The information for the A-ASSOCIATE.REQUEST, P-CONNECT.REQUEST, and S-CONNECT.REQUEST can be transmitted in a single protocol data unit (PDU). The T-CONNECT.REQUEST requires a separate PDU as the OSI protocols forbid the transport service user (i.e., the session service) from putting any of its own connection establishment information in the T-CONNECT.REQUEST PDU. Similarly, the responder places the T-CONNECT.RESPONSE in a single PDU and returns the S-CONNECT.RESPONSE, P-CONNECT.RESPONSE, and A-ASSOCIATE.RESPONSE in another PDU.

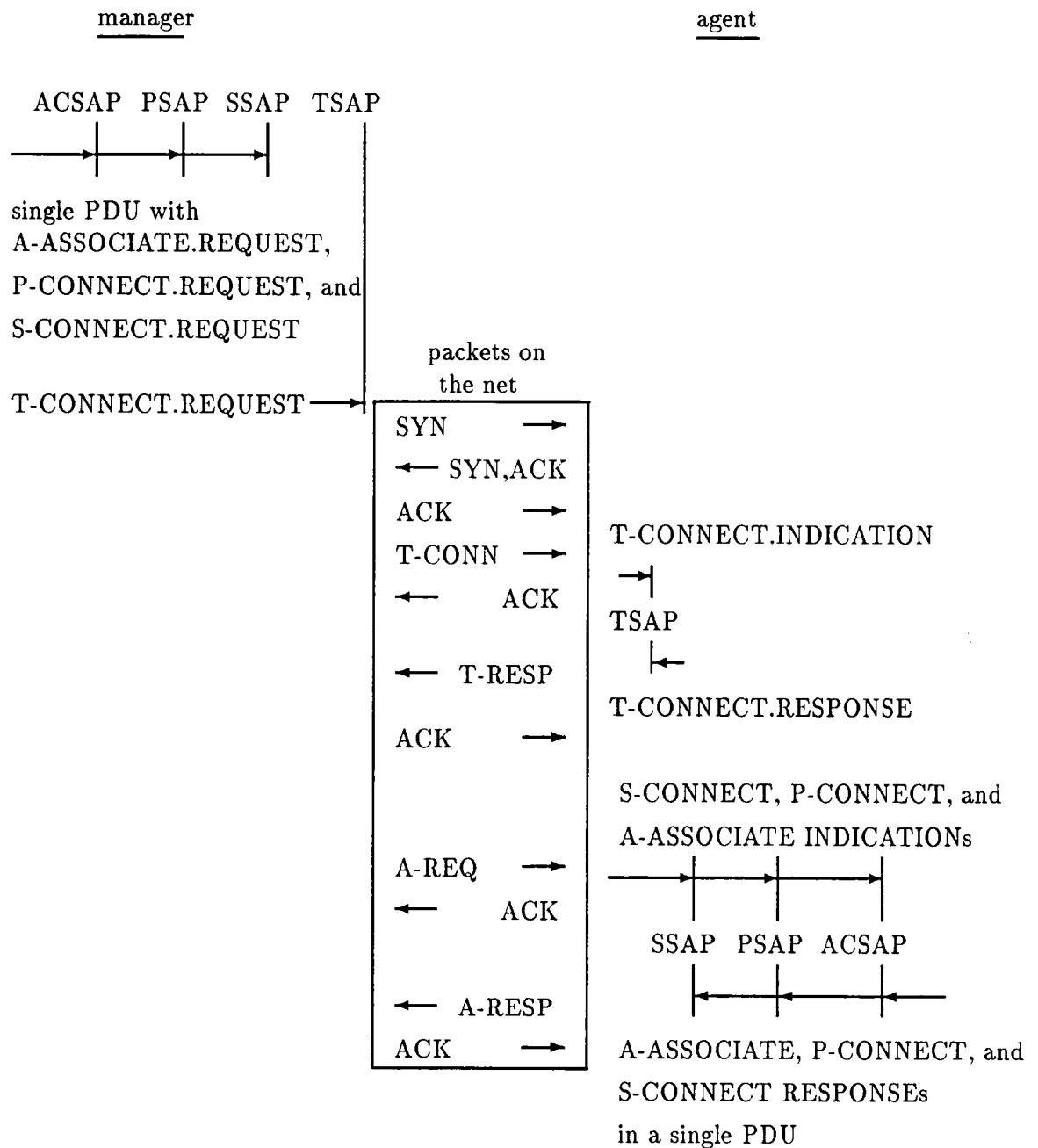


Figure 4.2: The association establishment phase of CMIP — a connection-oriented management protocol.

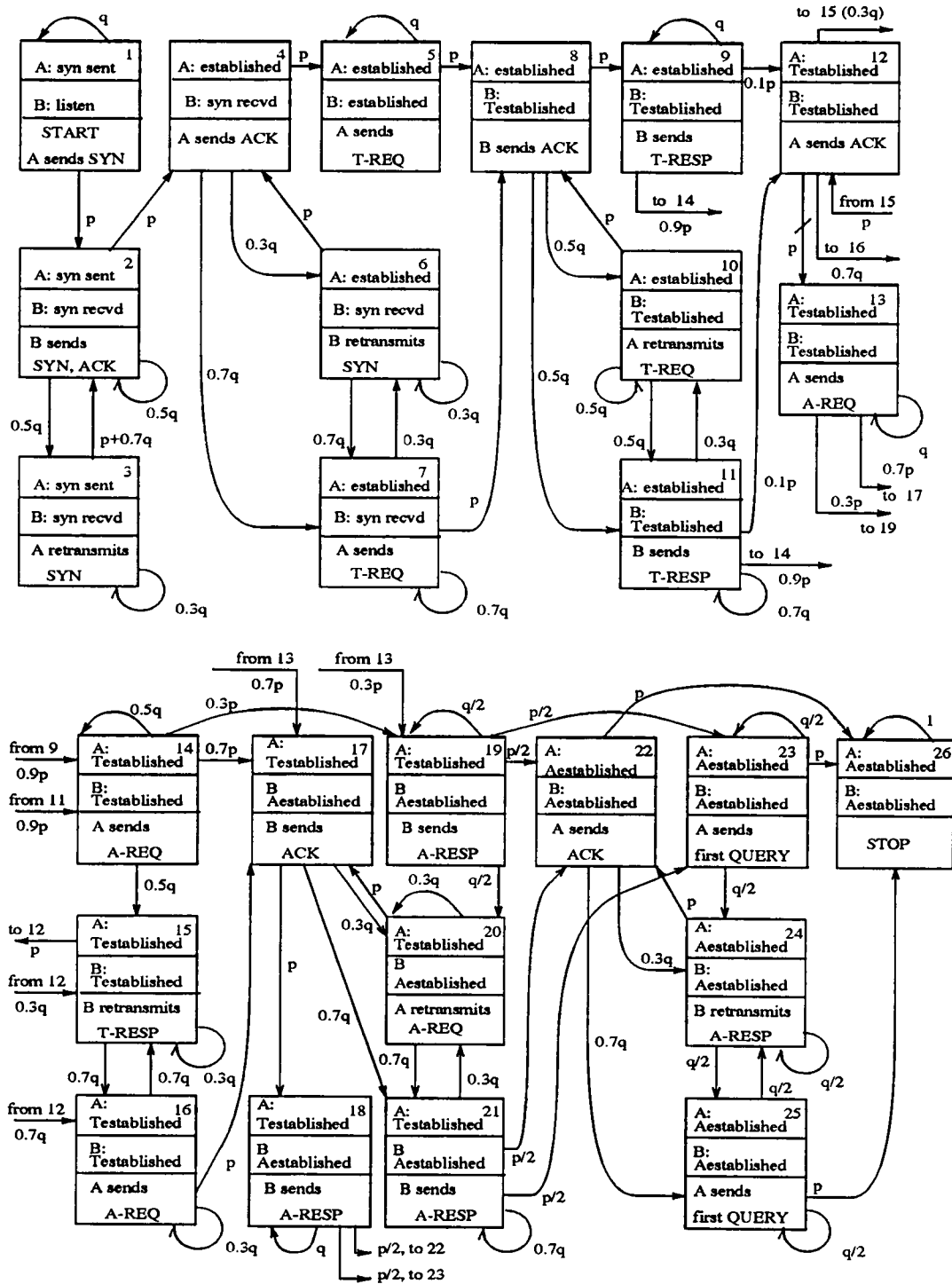


Figure 4.3: A Markov model for the association establishment phase.

A detailed explanation of the Markov chain that represents connection establishment over a lossy network follows. Each state in the process is completely determined by four items of information.

1. The state of the initiating entity.
2. The state of the responding entity. The possible states are:
 - (a) **listen**: The passive TCP is listening for a new connection.
 - (b) **syn sent**: The first SYN of three-way handshake has been sent, an ACK, SYN is awaited. Only the active TCP may be in this state.
 - (c) **syn recvd**: A SYN was received and a SYN-ACK has been returned. An ACK or data packet containing the ACK is awaited. Only the passive TCP may be in this state.
 - (d) **established**: This node has moved to TCP's ESTABLISHED state. The initiating TCP does this when it receives the ACK, SYN in response to its own initial SYN. The responding TCP does this when it receives an ACK or data following its ACK, SYN.
 - (e) **Testablished**: The transport service (TS) views the connection as established. The initiating TS does this when it receives the T-CONNECT.RESPONSE (T-RESP) PDU. The responding TS enters this state when it dispatches the T-RESP PDU in response to a T-CONNECT.REQUEST (T-REQ) PDU.
 - (f) **Aestablished**: The association control service (ACS) views the connection as established. The initiating ACS does this when it receives the A-ASSOCIATE.RESPONSE (A-RESP) PDU. The responding entity moves to this state when it dispatches the A-RESP PDU.
3. The packet being transmitted at this state.
4. The arcs coming into or leaving the state.

The arcs marked with a p or a fraction thereof, like $0.9p$ indicate the transition that would occur if the packet were to reach its destination. On occasion, any of several states may be entered when a packet reaches its destination. Such transitions are marked by fractions of p , like $0.1p$ or $0.9p$, depending on the relative probabilities with which those states may be reached. Similarly, transitions that occur when a packet is dropped are marked by q . If one of several states may be reached when a packet is

dropped, the corresponding transition arcs are labeled by fractions of q , like $0.7q, 0.3q$, etc. The sum of the probabilities on all the arcs leaving a state must equal $p + q = 1$.

In Figure 4.3, node A is the manager or the active transport service (TS), and B is the passive TS that listens for an incoming connection request. The rationale behind a few selected transitions will be provided in order to help the reader follow the model. Consider the transition out of state s_2 . If the SYN-ACK reaches, then station A will transmit an ACK. However, two things may happen if the packet is dropped. The process may go to s_3 , since A's retransmission timer is now on, it may retransmit the original SYN segment for B's acknowledgement has been dropped. On the other hand, B has also started its retransmission timer after dispatching its SYN-ACK segment, so this segment may also be retransmitted. The method of assigning probabilities to such competing transitions has been explained in a subsequent section.

It may seem that since A's retransmission timer started before B's, A will retransmit first. This is not necessarily the case since the timers on the two nodes are not synchronized, and their estimates of the round trip time may vary. Besides, the exact moment of retransmission will also depend on the other load being supported by the manager and agent nodes. Additional load will slow down the transport service and thereby delay the retransmission.

The transitions out of state s_3 are also instructive. Note that if A's retransmitted SYN reaches B, the process will move to s_2 where B sends another SYN-ACK. However, if A's retransmission is lost, the process could still move to s_2 if B's retransmission timer on the SYN-ACK PDU expires, or A might decide to retransmit the SYN if its own retransmission timer expires first. The latter case is indicated by a self loop.

It might help to note that a packet that is retransmitted is the same as the one that was first sent or retransmitted in a state located two arcs up the graph (except for self loops). This is logical as two arcs may be interpreted as the approximate round-trip time estimated by a node, which retransmits when this time expires.

The possible transition from s_9 to s_{14} without going through s_{12} may occur because in most cases the acknowledgement being sent in s_{12} was piggybacked on the A-REQ PDU being sent in s_{14} . Similarly, though a direct transition from s_7 to s_9 is possible, such a transition has not been observed in practice, hence it is not part of the model. Such special characteristics of a particular environment can only be detected by monitoring a packet trace of the activity. State s_{26} is the absorbing state. Both nodes reach the Aestablished state at this point. All the transitions in the model can be explained by reference to Figure 4.2 and by considering which retransmission timers are active at a given moment.

Assigning Probabilities to the State Transitions

The values of p and q are assumed to be given. Hence, if the delivery or loss of a packet results in only one possible transition in each case, they can be assigned probabilities of p and q directly. If there is more than one possible transition when a packet is delivered, then the probability p should be distributed among these possible transitions. Similarly, if there is more than one possible transition when a packet is dropped, the probability q has to be distributed among the possible transitions. The method used for this distribution is the subject of this section.

For example, consider the transitions from state s_4 in Figure 4.3. Note that when the packet transmitted by the process in this state is dropped, the process may move to s_6 or s_7 . In order to determine $p_{4,6}$ and $p_{4,7}$ the process is first driven to s_4 . The packet transmitted by A at this state is dropped by a mechanism that can drop selected packets. One can then observe the frequency with which the process moves to s_6 and s_7 over repeated trials. If $x\%$ of the trials move to s_6 , then one would assign a probability of $xq/100$ to that transition arc and a probability of $(1 - x)q/100$ to $p_{4,7}$. However, observing only the first transition from s_4 would give the proper results only if $p_{4,6}$ and $p_{4,7}$ are known to be constant over all transitions from state s_4 .

Such an assumption is not strictly true as a state transition will often depend on the times at which the retransmission timers on either side expire. The timers are however independent and the times at which they expire are unrelated. This will lead to an unpredictable variation in $p_{i,j}$ every time the process enters s_i . The process was driven to s_i to obtain estimates of the transition probabilities under these circumstances. The frequency of transition to the subsequent states was observed over several such trials in order to assign the probabilities. Although the tools exist to drop selected packets, it is still not possible to drive the process repeatedly to a state, especially if the probability of entering that state is low. Fortunately, small errors in apportioning the probabilities do not affect the final results greatly, hence it is enough to assign estimates in order to get fairly accurate results. Most of the probabilities assigned in the model are only indicative of a bias towards a particular transition rather than an accurate representation of the actual probabilities.

Validating the Model

A facility for dropping random packets has been built by modifying the Ethernet device driver on the agent node. A uniform random number generator is used to drop a certain percentage of packets that are transmitted by the agent port or are addressed to the

agent port. The model presented here has been constructed from prior knowledge of the behavior of the TCP protocol and the ISO association establishment mechanism. The behavior of the system while operating over a lossy net was studied with the help of the packet dropping mechanism. The model is largely based on this study. It is further validated by comparing its predictions with experimental results. Table 4.1 provides the predicted and experimental means of the number of packets required to complete CMIP's association establishment phase under different rates of packet loss.

The experiment requires the manager to issue the following commands:

connect <agent>

get <some_object>

The **connect** command sets up an association at the application layer. A **get** command is used immediately after, for if the final ACK generated in state s_{22} of the model is dropped, this query (it could be any query) would piggyback the ACK as shown in state s_{25} . This expedites connection establishment, especially during periods of high packet loss.

An association is completely established when either an A-ASSOCIATE.RESPONSE (A-RESP) and the corresponding ACK reach the manager and the agent respectively, or when the first query reaches the agent.

An association will fail when a segment requires frequent retransmission. The association is closed unilaterally when the number of retransmissions reaches a predefined threshold. An association is considered a failure as soon as the manager detects the loss of connection. In order to measure the time and packets transmitted before a connection loss occurs, it is necessary to define when a manager detects a loss of connection. The following points address this issue.

- If a TCP connection attempt times out before it reaches the ESTABLISHED state, the connection is closed quietly. If this happens on the manager's side, the manager emits a warning message. The time and number of packets transmitted before this warning is issued may be measured as necessary. If this happens on the agent's side, the manager can only learn of the closure when it attempts to send a packet and receives a reset in return. From the manager's perspective, this reset would mark the end of the connection.
- If the connection times out while TCP is in the ESTABLISHED state, a reset is sent to the peer. A connection may therefore be considered closed when the manager transmits or receives such a reset.

Table 4.1 reports the theoretical and experimental results for $q = 0.1$ to $q = 0.5$. Equation 5.1 has been used to determine that the sample means have a confidence interval of 10% at a 95% confidence level. Fifty readings have been taken in each case, this is more than what is required to achieve an accuracy of $\pm 10\%$. However, the additional readings serve to give us a better estimate of the experimental standard deviation. The observed standard deviations show good agreement with the predicted values. However, the accuracy of the observed standard deviations could not be determined statistically, since no method for doing this for non-normal samples is known to the author. The theoretical and experimental data are in close agreement until $q = 0.5$. Beyond this point, it is not possible to match the sample means with the predictions, since for higher values of q , the number of dropped connections will predominate. This would violate one of the principal assumptions on which the models are based. The data gathered for values of $q > 0.5$ is analyzed separately in order to calculate the MNPC.

Table 4.1: Observed and theoretical data for the association establishment phase.

q (drop rate)	packet requirements				observed time	
	theoretical		experimental		min:sec	
	mean	sd	mean	sd	mean	sd
0.0	9.80	1.18	9.68	0.55 ¹	0:01	0:00
0.1	11.14	1.78	11.00	1.61	0:03	0:04
0.2	12.80	2.39	12.50	2.07	0:08	0:10
0.3	14.87	3.08	14.82	4.19	0:42	1:16
0.4	17.53	3.93	16.34 ²	4.33	1:04	1:56
0.5	21.08	5.01	19.62 ³	5.25	2:07	2:20

With reference to the theoretical results, it is seen that the mean total number of packets required to complete the transaction rises with increasing packet drop rate. This is expected. However, the standard deviation (sd) also increases. This is also intuitively explained. As the net starts to lose more and more packets, the process becomes more unpredictable, as the exact sequence of packets lost affects the total

¹When $q = 0$, the number of times the process is in each transient state can no longer be assumed to be independent. The standard deviation predicted by the model is therefore not valid for $q = 0$.

²Two connections out of fifty two attempts timed out. Only the successful connections are used to calculate the sample mean and standard deviation.

³Seven connections out of fifty seven attempts timed out. Only the successful connections are used to calculate the sample mean and standard deviation.

number of packets required to complete the process. This sequence will change over different observations. Hence with any fixed value of q , the total number of packets required to complete the association will vary widely over different experiments. Since this variation is intrinsic in the process itself, no method can provide theoretical means with a uniformly small variance.

Note that connections were lost due to excessive packet loss during the three way handshake for $q = 0.4$ and $q = 0.5$. Connections may also be lost due to excessive retransmission of any data segment. Because the observations where the connection was reset had to be discarded, the observed mean may be somewhat lower than the predicted mean, especially for higher values of q , when many connections are expected to timeout. The models introduced in this chapter do not handle the possibility of the connection timing out.

One may expect that since 10 packets are usually required to complete the association when $p = 1$, the number of packets required to complete the transaction when $p < 1$ will be $10/p$, since if $10/p$ packets are transmitted, 10 may be expected to reach their destination. On closer examination, it was found that this may not be the case. When packets are dropped, more than 10 packets are required to be transmitted successfully in order to establish the connection. This is because once packets start being dropped, each successfully transmitted packet may no longer take the process closer to its goal than where it was before the failure occurred. In the model of Figure 4.3, a dropped packet may bring the process to s_6 from s_4 . A packet has to be successfully delivered now for the process to move back to s_4 . However, this successful transmission does not take the process any closer to completion than it was when it first entered s_4 . Such situations increase the number of successful transitions that are required to complete the transaction in a lossy network. If the number of successful transmissions required could be calculated as T , then the total number of transmissions could be calculated as T/p . However, T has no relation to the 10 packets required to complete the transaction in a lossless net, it is a random number depending on the packets dropped and the states the process moves in to. T will vary from experiment to experiment even if q is fixed.

On the other hand, it is also possible to drop ACK packets and yet not pay a penalty, because subsequent data packets will piggyback the ACK. This is illustrated in the transitions from s_4 to s_7 , s_8 to s_{11} , s_{12} to s_{16} , and s_{17} to s_{21} . Hence if the ACKs transmitted in states s_4 , s_8 , s_{12} , or s_{17} are lost, the process could still take only 10 packets to complete.

Association Establishment Under High Loss

Since the models do not account for a connection timing out due to excessive retransmission, they cannot predict the behavior of the association establishment phase for values of q greater than 0.5. These situations will be dealt with experimentally.

Tables 4.2 to 4.4 provide data from observations where q exceeds 0.5. Fifty observations were made in each case. In these tables a * indicates that a connection timed out during the observation. All times have been shown in minutes and seconds, i.e., mm:ss. In addition to the experimental data, the following terminology will be necessary to calculate the total number of packets required to establish an association. Here a successful trial indicates that an association could be established. Failure indicates that a connection was timed out.

- t : total number of trials
- s : number of successful trials
- r : probability of success
- x : number of trials up to and including the first success
- PS_{avg} : average number of packets required during a successful trial
- PF_{avg} : average number of packets exchanged before a failure
- TS_{avg} : average time required during a successful trial
- TF_{avg} : average time before a failure
- P_A : average total number of packets required to establish an association
- T_A : average total time required to establish an association

The following calculations are based on the data in Table 4.2. Note that PS_{avg} was calculated by summing the packets required during successful trials and then dividing by the total number of successful trials. Table 4.5 shows the final results for the high values of q . All 50 attempts at connection establishment failed with $q = 0.9$.

$$q = 0.6, \quad t = 50, \quad s = 30$$

$$r = \frac{s}{t} = \frac{30}{50} = 0.60$$

$$x = \frac{1}{r} = 1.67$$

$$PS_{avg} = 21.30, \quad PF_{avg} = 15.30$$

$$TS_{avg} = 3 : 57, \quad TF_{avg} = 5 : 18$$

$$P_A = (x - 1)PF_{avg} + PS_{avg} = 31.55$$

$$T_A = (x - 1)TF_{avg} + TS_{avg} = 7 : 30$$

Table 4.2: The number of packets and the time required to complete association establishment, $q = 0.6$.

27 8:56	12* 1:57	18 1:56	24 5:56	4* 1:15	25 4:51	25* 11:06	12 0:08	26 7:40	41* 13:29
21 3:40	23 7:56	21 3:42	18 1:50	22 2:09	26 6:52	15 0:38	28 8:25	15 0:50	4* 1:15
35* 11:08	10* 1:16	17 0:10	4* 1:16	20 2:49	4* 1:16	11* 1:21	16 0:40	30 7:09	11 0:09
16 0:15	37 14:31	25 5:53	9* 1:15	8* 1:15	4* 1:16	20 1:33	22* 11:13	19 4:23	8* 1:16
9* 1:16	14 0:09	4* 1:15	19 0:53	27 9:48	26* 11:27	19 0:50	38* 19:23	28 4:05	28* 11:18

Table 4.3: The number of packets and the time required to complete association establishment, $q = 0.7$.

25 2:13	4* 1:15	4* 1:16	29 7:05	4* 1:15	17 0:24	4* 1:15	9* 1:15	9 1:16	24 1:19
7* 1:15	9* 1:15	9* 1:15	26 7:18	9* 1:16	7* 1:16	8* 1:15	32 7:59	4* 1:15	35* 11:09
4* 1:14	27 5:00	21 4:40	33 8:28	35 9:00	22 0:50	4* 1:15	31* 9:10	35* 18:31	37* 11:31
8* 1:16	9* 1:16	22 3:13	22* 8:32	28 6:58	7* 1:16	33 9:13	4* 1:16	19 0:59	4* 1:16
19* 6:56	30 9:15	17 0:57	4* 1:15	9* 1:15	11* 1:15	9* 1:15	4* 1:15	27 3:16	37* 8:41

Table 4.4: The number of packets and the time required to complete association establishment, $q = 0.8$.

36	4*	4*	8*	4*	14*	4*	34	22*	35
10:07	1:16	1:15	1:16	1:15	1:15	1:15	9:23	11:11	10:39
4*	9*	27	4*	4*	4*	9*	31*	32*	7*
1:15	1:15	7:04	1:15	1:15	1:15	1:15	11:39	12:38	1:16
8*	38*	4*	4*	4*	4*	17*	8*	4*	9*
1:15	15:26	1:17	1:16	1:15	1:15	11:06	1:16	1:15	1:16
4*	22*	4*	23*	4*	30*	4*	23*	4*	8*
1:16	11:11	1:16	11:12	1:16	11:17	1:15	8:30	1:16	1:16
19*	4*	4*	23*	8*	9*	12*	4*	14*	4*
11:06	1:15	1:16	11:07	1:16	1:15	1:57	1:15	2:39	1:15

Table 4.5: P_A and T_A values for various values of $q > 0.5$.

q	x	PS_{avg} (packets)	PF_{avg} (packets)	TS_{avg} (min:sec)	TF_{avg} (min:sec)	P_A (packets)	T_A (min:sec)
0.6	1.67	21.30	15.30	3:57	5:18	31.55	7:30
0.7	2.63	25.10	11.97	4:42	3:22	44.61	10:11
0.8	12.50	33.00	10.67	9:18	3:44	155.70	52:14

4.3.2 The Data Transfer Phase (One-Query-One-Object)

The Markov process for the data transfer phase is shown in Figure 4.4. It is assumed that each query and response fits in a single packet. A is the manager and B is the agent. As before, each state of the process is uniquely distinguished by four factors:

1. The state of the manager node (A). This could be one of the following:
 - (a) **Qsent**: The manager has sent a query, and is awaiting an ACK or response.
 - (b) **Arecvd**: The manager's TCP has received an acknowledgement to the query.
 - (c) **Rrecvd**: The manager has received a response to its query.
2. The state of the agent node (B). This could be one of the following:
 - (a) **listen**: The agent is listening for a query.
 - (b) **Qrecvd**: The agent has received a query.
 - (c) **Rsent**: A response to the query has been sent.
3. The packet being transmitted in this state.
4. The arcs coming into or leaving the state.

The state transitions are derived by noting that each management query is followed by an acknowledgement (ACK) and a response, or just a response (which would piggyback the ACK). Each response from the agent will be followed by an ACK from the manager and a new query. The acknowledgement to the previous response may be piggybacked by the new query. Retransmissions must also be accounted for. The starting state is s_1 . COUNT is a counter for the number of responses that have been received by the manager. It is set to zero when the process starts. The process moves to s_2 when a query is successfully transmitted. If the ACK sent by the agent reaches the manager, nothing special happens, the agent just sends out the response after a certain interval (s_5). However, if the ACK is dropped, one of two events could occur — the manager could retransmit the query (s_3), or the agent might transmit the response (s_4). Assuming that the query retransmitted in s_3 is sent successfully, one of two things might happen. The agent may send an ACK (s_2), or it may directly send the response (s_4). Similarly, when the retransmitted query is dropped, it may either be retransmitted again (s_3), or the agent might send a response (s_4), since it is already in receipt of the query.

Successful transmission from s_4 and s_5 are of particular significance, for this is where the manager receives the response to its query. Note that these transitions pass through the counter box (s_6). The function of s_6 is to keep a count of the number of responses that have reached the manager. Once this count reaches n , n being the total number of queries and hence the total number of responses, the process must transit from s_6 to s_{10} , on all other occasions it transits to s_1 or s_7 . Note that these transitions are not stochastic, since the process should definitely move to the absorbing state s_{10} when COUNT reaches n . However, to fit this requirement into the framework of a Markov model, one notes that out of n passages through s_6 , the process should move to s_{10} only once (when COUNT = n), and to s_1 or s_7 the other $(n - 1)$ times. Hence $p_{6,10}$ can be assigned a value of $1/n$, whereas $p_{6,1}$ and $p_{6,7}$ can each be assigned $(n - 1)/2n$. Such a model will be useful for calculating the total mean number of times the process is in the transient states. The variance however, will not be preserved.

If an ACK sent in s_7 reaches the manager, a new query is transmitted after some time as shown in s_1 . This new query may be transmitted even if the ACK does not reach the agent (s_8). However, if the ACK is lost, it is also possible for the agent to retransmit the response (s_9), since its retransmit timer has not yet been turned off by an acknowledgement. The remaining transitions are explained in a similar fashion.

One result of placing the decision box before s_7 is that the model is thereby incapable of accounting for the packets that will be transmitted after the final (n th) response reaches the manager. These packets include the final ACK from the manager, and the packets that might be retransmitted if this final ACK is lost. However if n is even moderately large ($n \geq 10$), this discrepancy will be negligible. The probabilities on the transition arcs have been assigned according to the methods discussed previously.

Validating the Model

The model for the data transfer phase is validated by comparing the predicted results with experimental results. Note that the variances predicted by the model will be much larger than those obtained experimentally. This is caused by the decision box or s_6 . Since according to the model, the process can exit to the absorbing state at any point before all the responses have been obtained, the variance in the number of packets that will be transmitted before the process is absorbed goes up. This variance will increase with the number of queries pumped into the system, i.e., with n . However, in reality, the process will end only when all n responses have been received. The experimental variance will therefore not be as high as the predicted variance. The total number of queries is 10. Queries requiring different execution times on the agent were selected,

since this has some bearing on whether an ACK will be sent after a query or whether the response will be sent directly, the ACK being piggybacked. Table 4.6 provides the theoretical and experimental values for the mean and standard deviations of the total number of packets needed to obtain responses to 10 queries ($n = 10$). The observed times are also given. The theoretical values of the standard deviation are of no practical value due to reasons mentioned previously. They are given here as they might promote a greater understanding of the model. The model is not valid for $q > 0.5$ as connection timeouts predominate.

Although less than 50 observations are required to obtain a 10% confidence interval on the means at the 95% confidence level, 50 observations were nevertheless taken in order to obtain an approximation of the experimental standard deviation. Increasing the number of observations to 50 has the effect of narrowing the confidence interval for the experimental means to $\pm 5\%$ at the 95% confidence level.

Table 4.6: Theoretical and experimental data for one-query-one-object transactions.

q (drop rate)	theoretical		experimental (50 observations)		time to completion min:sec	
	mean	sd	mean	sd	mean	sd
0.0	31.50	14.70	32.22	2.32	0:01	0:00
0.1	34.83	15.82	34.74	3.36	0:04	0:04
0.2	39.05	17.20	38.60	4.20	0:09	0:09
0.3	44.56	19.07	44.40	5.91	1:04	1:37
0.4	52.03	21.70	53.40	7.77	2:29	3:01
0.5	62.66	25.65	62.00 ⁴	9.96	6:01	3:58

Data Transfer (One-Query-One-Object) Under High Loss

Since the model does not account for a connection timing out due to excessive retransmissions, it cannot predict the behavior of the data transfer phase for values of q greater than 0.5. These situations will be dealt with experimentally.

Let it be assumed that 10 queries will constitute a complete management session. The number of packets and the time required to receive 10 responses are computed from the experimental data. The number 10 is of little significance, it is a compromise between a trivially small value and a value so large that experimentation over a lossy

⁴Thirteen connections were timed out during sixty three attempts at data transfer. Only the experiments that completed successfully were used to calculate the sample mean and standard deviation.

net would be prohibitively time consuming. Tables 4.7 to 4.9 provide data from observations when q exceeds 0.5. Fifty observations were taken in each case. The data include the number of packets, number of responses received, and the time elapsed before completion or timeout of the data transfer phase. The notation p/r means that p packets were exchanged and r responses were received. A value of r less than 10 indicates that the association timed out during data transfer. Times are in minutes and seconds, i.e., mm:ss. These data represent the data transfer phase only and does not include the association establishment phase. The case for $q = 0.9$ has not been considered as no associations could be established at this rate of packet loss. In addition to the data, the following additional terminology will also be necessary for subsequent computation. The values of P_A and T_A can be obtained from Table 4.5. Table 4.10 lists the values of P_{total} and T_{total} . The P_A and T_A values shown there are from Table 4.5.

- R : average number of responses received before the association times out
 - P_D : average number of packets required in the data transfer phase before the connection is lost
 - T_D : average time spent in the data transfer phase before the connection is lost
 - C : number of associations required before all 10 responses are obtained
 - P_{total} : sum total of the number of packets required to receive 10 responses
 - T_{total} : total time elapsed before responses to all queries are received
- P_{total} and T_{total} can now be computed as follows:

$$C = \frac{10}{R}$$

$$P_{total} = P_A C + P_D C \quad (4.1)$$

$$T_{total} = T_A C + T_D C \quad (4.2)$$

Equation 5.1 shows that the confidence interval of the sample mean decreases when the sample size is increased, the other parameters being constant. Faced with this situation, it was felt that a wider margin of error was acceptable if that meant that the sample size and therefore the time required to conduct an experiment could be kept within reasonable limits. The numbers shown in Table 4.10 have relatively wide confidence intervals at a 90% confidence level since a number of experimental observations have been combined to arrive at these figures. This is acceptable, since the aim here is to make a comparative study of the behavior of the connection-oriented and the connectionless modes of network management and to observe any variation in the behavior of the protocols with varying rates of packet loss.

Table 4.7: Observed data for one-query-one-object transactions, $q = 0.6$.

66/10 7:29	65/10 6:59	23/2 9:05	68/10 10:07	39/3 13:16	51/5 9:12	57/7 9:46	37/4 8:39	37/3 10:07	46/5 10:41
38/3 9:24	33/3 8:31	62/7 11:51	48/6 10:59	47/4 9:04	75/10 14:05	38/3 9:05	46/5 9:41	44/6 9:42	37/4 10:28
27/2 9:52	33/1 8:31	64/7 18:19	41/5 11:57	31/3 8:42	40/3 8:38	46/6 9:08	46/5 9:29	53/8 9:52	43/3 8:39
75/10 8:40	66/10 7:27	67/10 15:38	70/8 15:40	65/10 6:57	58/10 4:52	70/8 18:57	41/4 9:11	20/1 8:34	71/7 22:27
39/2 9:04	73/10 4:05	80/10 15:06	65/10 7:23	39/10 0:17	53/7 10:39	30/3 8:31	72/10 14:34	71/7 17:16	34/3 8:33

Table 4.8: Observed data for one-query-one-object transactions, $q = 0.7$.

39/2 9:05	29/2 9:02	86/6 22:56	36/2 9:36	43/3 13:14	32/2 9:04	51/6 10:02	63/5 17:01	65/10 6:06	33/1 8:31
40/3 9:41	41/4 9:06	44/4 9:35	42/4 8:31	38/3 9:33	42/4 9:34	37/2 9:05	39/3 9:50	33/2 8:47	35/3 8:47
62/6 13:36	39/4 9:34	37/3 9:38	41/4 9:08	51/5 11:45	37/3 10:42	73/10 10:43	31/2 9:28	29/1 8:33	44/3 16:20
71/10 6:27	40/3 15:07	24/1 8:33	60/10 5:35	36/3 9:03	27/1 9:03	38/2 9:39	33/3 8:35	46/5 9:35	30/1 9:34
83/10 13:56	72/10 10:55	23/1 8:37	40/3 8:31	25/1 8:39	26/1 8:34	28/1 12:09	33/1 9:06	68/5 14:42	83/10 16:47

Table 4.9: Observed data for one-query-one-object transactions, $q = 0.8$.

25/1 9:02	37/2 9:49	41/5 11:45	34/2 9:42	30/2 8:32	29/1 8:32	62/7 11:51	35/2 9:16	54/3 12:49	17/1 8:31
31/1 9:02	30/1 8:33	33/1 8:34	36/2 9:03	33/3 9:17	29/1 8:33	30/1 8:31	34/2 8:35	30/1 9:01	31/1 14:41
33/1 8:33	28/1 8:35	39/3 10:15	39/2 9:38	31/1 9:01	40/3 8:45	38/2 9:35	35/1 9:18	32/2 8:32	32/1 8:30
30/1 8:31	28/1 9:02	29/1 9:03	30/1 8:33	27/1 8:31	23/1 8:31	29/1 8:30	32/1 9:11	25/1 8:33	27/1 8:46
47/4 9:04	14/0 8:31	34/1 9:03	46/6 9:08	25/1 8:38	33/1 8:46	26/1 10:05	65/10 6:56	29/0 8:31	34/2 9:39

Table 4.10: The P_{total} and T_{total} values for $q = 0.6$ to 0.8 .

q	R	C	P_A (packets)	T_A (mins:secs)	P_D (packets)	T_D (mins:secs)	P_{total} (packets)	T_{total} (hrs:mins:secs)
0.6	6.06	1.65	31.55	7:30	50.80	10:18	135.88	00:29:22
0.7	3.88	2.58	44.61	10:11	43.96	10:23	228.51	00:53:03
0.8	1.50	6.67	155.70	52:14	31.90	9:14	1251.29	06:49:56

The results of Table 4.10 support the following conclusions:

1. P_{total} and T_{total} vary sharply with q , the rate of packet loss. This is clear from the sharp increase in their values as q increases from 0.7 to 0.8.
2. When multiple management operations are required to be performed, it may be necessary to go through a number of association establishment phases before all the operations can be completed. It is clear that with increasing q , the number of packets and the time required for the association establishment phase outstrips the number of packets and the time spent in the data transfer phase. The total time required to complete all management operations increases to the extent where effective network management is no longer possible. **The connection-oriented mode of network management therefore does not provide satisfactory service during periods of high packet loss.**

4.3.3 On Separating the Association Establishment and the Data Transfer Phases

Separating the association establishment and data transfer phases has allowed the construction of separate models for the two stages, leading to ease of understanding and analysis. However, though it is convenient to consider them separately, there is a link between them, disregard of which introduces an error in the preceding simplified analysis. The purpose of this section is to explain the source of the error and provide an upper bound for this error.

The Markov chain model for the association establishment phase includes a stage where the manager sends the first query. The model reached the absorbing state as soon as this query reached the agent. Notice that this sequence of events is also repeated at the beginning of the model for the data transfer phase. Therefore the sequence of events from when the first query leaves the manager to the time it first reaches the manager is duplicated in the two models. The values of P_{total} and T_{total} therefore undergo a

corresponding increase. The increase in the number of packets due to this overlap can be written as $\frac{C}{p}$, i.e., the expected error in the value of P_{total} is $\frac{C}{p}$.

This error is produced because the probability of the first query reaching the agent is p , therefore $\frac{1}{p}$ transmissions of the first query are expected before it reaches the agent the first time, thus signaling the end of the association establishment phase. Therefore an expected overlap of $\frac{1}{p}$ packets will occur between the two phases, each of the C times the data transfer phase is entered. Therefore the total expected overlap or error is $\frac{C}{p}$ packets.

The expected error in the value of T_{total} requires an understanding of the TCP retransmission mechanism as well as an assumption about the value of the round trip estimate when the first query is transmitted initially from the agent. Note that BSD TCP uses a table of multiplying factors to determine the retransmission timeout (RTO) in the absence of acknowledgements, the current round trip time estimate being used as an approximation of the initial RTO. This table is of the form ⁵:

(1, 2, 4, 8, 16, 32, 64, 64, 64, 64, 64, 64, 64)

The RTO for successive retransmissions of a packet therefore doubles at each step before leveling out. Since $\frac{1}{p}$ transmissions of the first query are expected before it reaches the agent, the preceding table will be used $(\frac{1}{p} - 1)$ times to determine the RTO before each retransmission.

The expected total time for retransmission of the first query before it reaches the agent is therefore

$$T_{overlap} = RTO_{initial}(1 + 2 + 4 + \dots + (\frac{1}{p} - 1)^{th} \text{ term})$$

The expression $(\frac{1}{p} - 1)$ is to be rounded off the nearest integer before performing the calculation. The terms in the series are from the preceding table. If $p = 0.2$ and $RTO_{initial} = 1$ second, the value of $T_{overlap}$ will equal 15 seconds. The value of the overlap will be smaller for higher values of p .

NOTE: The initial RTO is identical to the current estimate of the round trip time. The estimate of the initial RTO used above is reasonable because the association establishment phase preceding the data transfer phase was conducted under conditions of no loss, therefore the round trip time estimate at the time the first query is dispatched is smaller than 1 second. In the local TCP, if it is smaller than 1 second, it is rounded up to 1 second.

⁵Obtained from SunOS 4.1.1 source code (/sys/netinet/tcp_timer.c)

4.3.4 A Simple Approximation of the Protocol's Packet Requirements

Table 4.11 presents the experimental means from Tables 4.1 and 4.6 along with values for $\frac{m}{p}$, where m is the experimental mean obtained for $p = 1$. As usual, p is the probability of a packet reaching its destination.

Table 4.11: Comparing the approximation $\frac{m}{p}$ with observed data.

p (pass rate)	association establishment ($m = 9.68$)		data transfer ($m = 32.22$)	
	experimental	$\frac{m}{p}$	experimental	$\frac{m}{p}$
0.9	11.00	10.76	34.74	35.78
0.8	12.50	12.10	38.60	40.28
0.7	14.82	13.83	44.40	46.03
0.6	16.34	16.13	53.40	53.70
0.5	19.62	19.36	62.00	64.44

The experimental means are closely tracked by the $\frac{m}{p}$ values. The expression $\frac{m}{p}$ can therefore serve as a simple and convenient approximation for the number of packets required by the management protocol at values of $p \geq 0.5$, below which frequent connection failures render the approximation invalid. This approximation can be explained by the following argument.

A connection-oriented transaction over a lossy net can be represented by the diagram in Figure 4.5. The arrows which do not reach the other side depict dropped packets. This is for higher values of p . As described later, the situation changes for lower p values.

The manager sends requests and the agent sends back the responses. Both requests and responses may be dropped as shown by the arrows which do not reach the other side. These packets require retransmission. It is important to assume that one side stops retransmitting as soon as its packet reaches the other side. In other words, a side does not retransmit a request or a response while the other side is attempting to transmit the corresponding response or the next request. This behavior characterizes protocol behavior for high values of p and is the basis of the $\frac{m}{p}$ approximation.

Under these circumstances, the total number of packets required to complete the transaction (until the cut off mark only) is given by

$$P = request1_{total} + response1_{total} + request2_{total} + response2_{total} + ack_{total} \quad (4.3)$$

where $request1_{total}$ is the total number of transmissions of request1, etc. The expected

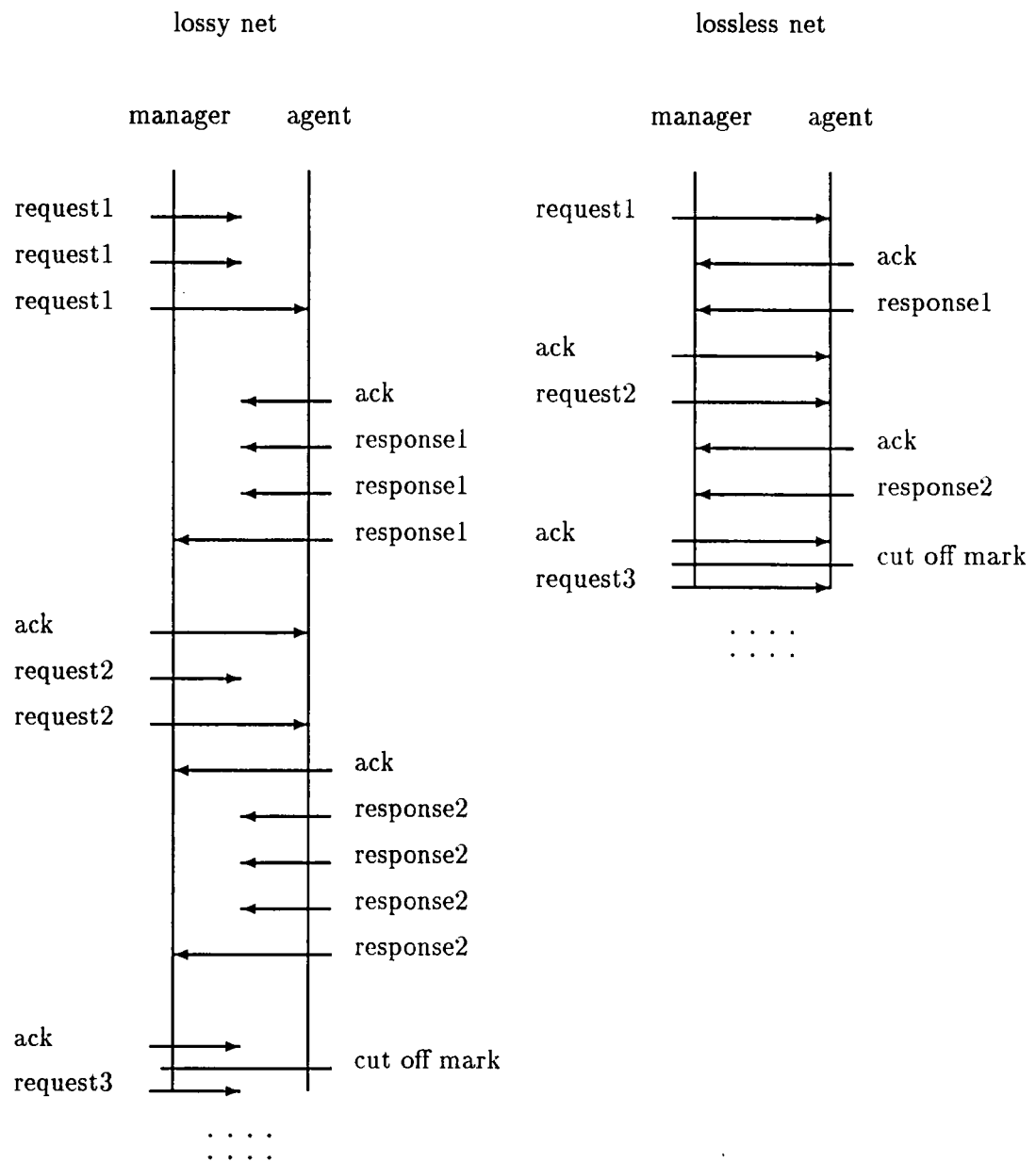


Figure 4.5: Connection-oriented transaction over a lossy net (left) and a lossless net (right).

value of $request1_{total}$, $response1_{total}$, $request2_{total}$, and $response2_{total}$ is given by $\frac{1}{p}$. Expression 4.3 can therefore be rewritten as

$$P = \frac{1}{p} + \frac{1}{p} + \frac{1}{p} + \frac{1}{p} + ack_{total} = \frac{4}{p} + ack_{total} \quad (4.4)$$

From Figure 4.5(right), we find that under conditions of no loss, the number of packets required to complete this transaction would be

$$\begin{aligned} m &= 4 + ack_{total} \\ \Rightarrow \frac{m}{p} &= \frac{4}{p} + \frac{ack_{total}}{p} = P + ack_{total}(\frac{1}{p} - 1) \end{aligned} \quad (4.5)$$

Comparing Equations 4.4 and 4.5, and assuming that the number of acknowledgement packets is small compared to the total number of packets we can write

$$\frac{m}{p} \approx P \quad (4.6)$$

The amount error is of course

$$ack_{total}(\frac{1}{p} - 1)$$

The assumptions necessary for this approximation to hold are reiterated.

1. The number of acknowledgements issued during the transaction are small relative to the total number of packets used.
2. A data packet will not be retransmitted after it has reached the other side once.

Assumption 2 holds for the major part of a transaction when p has a high value. To understand why this is so, refer to Figure 4.5 once more. If the acknowledgement reaches its destination (i.e., the ACK to request2), this condition obviously holds true, since there is no possibility of request2 being retransmitted while the retransmission of response2 is in progress. At high p values it is expected that most acknowledgements will reach their destination. The matter becomes more complex when an acknowledgement is dropped (i.e., the ACK to request1). However, under certain conditions the assumption will still hold. When request1 reaches the agent, the RTO for this packet has already advanced due to the retransmissions. For this reason, when the agent starts transmitting response1, the RTO for response1 is much lower than the RTO for request1. Therefore at least one retransmission of response1 is possible before request1 is retransmitted. For high values of p , response1 will usually reach the manager within these first few retransmissions, thus cancelling the need for the manager to retransmit

a packet that has already reached the agent once. The same argument applies to all the data packets when the value of p is high. Observations show that a small proportion of the transmissions will violate this assumption. The extra packets generated by this violation are absorbed by the error term to some extent. The preponderance of connection failures makes this approximation inapplicable at low values of p ($< 50\%$).

4.3.5 The Data Transfer Phase (One-Query-Multiple-Objects)

Introduction to CMIP's Linked Replies

If the reply to a single request for a set of managed objects requires that more than one managed object instance be returned, these managed object instances will be returned as a series of separate PDUs. All the instances cannot be returned in a single PDU because the CMIP response PDU provides only a single field for the object instance information. A single response PDU can therefore contain information about a single managed object instance only. The response PDUs sent in reply to a single query are known as linked replies, as they are all associated with the same query.

The packet flow for linked replies is difficult to model due to the following reasons:

1. It is not known a-priori, how many packets will be used by the agent to transmit the entire response. The replies may be segmented into an arbitrary number of packets by the transport layer.
2. The agent can transmit a packet without waiting for an acknowledgement of the previous packet. This means that packets could be received out of order by the manager.
3. The transport protocol (TCP for example) can choose to concatenate segments that await retransmission into a bigger packet.
4. The problem of modeling connection failures remains as before.

The creation of a predictive tool for use with linked replies is therefore left as an open problem at this time.

Since linked replies are best suited to retrieve entire tables or selected entries of a single table, it will be studied here in that context only. The query used in these experiments will retrieve all entries in the `ipRoutingTable`.

Recall that when separate queries were used, managed objects that had not yet been retrieved could be requested after re-establishing a connection that is lost during data transfer. Therefore, the amount of information to be retrieved would decrease

gradually after every attempt to transfer data. The linked reply mechanism is an “all or nothing” arrangement, especially when elements from a single table are being retrieved, since CMIP provides no way of specifying a part of the table that follows the entry that was last successfully received. This is a drawback, for if the connection fails during data transfer, the entire transaction will have to be repeated. The probability of another failure before the transaction terminates successfully will therefore remain the same as before.

Data and Analysis

Table 4.12 provides the observed values for the mean and standard deviation of the total number of packets and the time required to retrieve all entries of the `ipRoutingTable` (10 in this experiment). The observations for higher values of q have been treated separately as the transport connection then starts to fail frequently.

Table 4.12: Observed data for one-query-one-object transactions.

q (drop rate)	packets		time (mm:ss)	
	mean	sd	mean	sd
0.0	8.04	0.99	0:01	
0.1	8.92	1.70	0:01	0:01
0.2	9.94	1.92	0:02	0:03
0.3	11.68 ⁶	3.19	0:18	0:52
0.4	13.34 ⁶	4.28	0:44	1:49
0.5	15.72 ⁷	4.96	1:35	2:29

Tables 4.13 to 4.15 provide the number of packets and the time required from observations where q exceeds 0.5. Fifty observations were taken in each case. A * indicates that the connection deadlocked during the observation. An X indicates that the connection timed out and the manager detected it. Times are in minutes and seconds, i.e., mm:ss. Calculations for the expected total number of packets required for the completion of data transfer follow. Linked replies are used here. The necessary terminology has been reiterated for ease of reference.

⁶One connection was timed out during fifty one data transfer phases. Only the experiments that completed successfully were used to calculate the sample mean and standard deviation.

⁷Two connections were timed out during fifty two data transfer phases. Only the experiments that completed successfully were used to calculate the sample mean and standard deviation.

Table 4.13: Observed data for one-query-multiple-object transactions, $q = 0.6$.

24*	26*	17	26*	11	17	26*	24*	11	13
8:37	8:33	3:42	8:32	0:11	1:03	8:54	8:32	0:05	0:10
19	22	29X	10	23*	35*	32*	21	25X	19
4:47	2:07	8:38	0:05	8:38	9:33	9:34	2:38	8:38	1:10
13	24	25	17	16	18	25*	13	15	15
4:26	7:29	5:20	0:48	0:31	0:22	8:30	0:04	1:02	0:14
25X	27*	14	31X	13	23	23	13	14	32*
8:30	8:33	0:29	9:02	0:05	6:23	2:39	0:22	0:06	11:42
28X	22	18	16	23	28*	23*	30*	24	29*
8:31	1:40	2:39	0:31	2:45	9:05	9:04	8:45	5:26	9:17

Table 4.14: Observed data for one-query-multiple-object transactions, $q = 0.7$.

16	27*	24X	25*	15	11	26X	26X	29	26*
1:03	9:33	8:37	8:37	1:03	0:02	8:38	8:31	9:34	8:31
15	30X	11	23*	27	19*	27X	28*	20*	20
0:25	9:17	0:05	8:34	4:45	8:31	9:02	8:45	9:03	1:09
28*	23	20	11	24*	25*	29X	24*	32*	25*
10:38	4:18	3:11	0:02	9:05	9:07	8:38	8:31	9:18	8:46
23*	27	25*	30*	11	23*	27*	25X	28X	24*
8:33	7:42	8:46	8:38	0:07	8:32	8:53	8:31	9:17	8:31
33X	26*	35X	25X	21*	18	35*	25*	23*	23
14:21	9:02	16:29	10:05	8:31	2:40	14:21	8:31	8:34	4:18

Table 4.15: Observed data for one-query-multiple-object transactions, $q = 0.8$.

9	25*	29*	28*	19	21X	21*	24*	25*	24*
0:01	9:02	8:31	10:37	2:39	9:05	8:31	8:46	8:34	8:41
15	27*	25X	27*	24*	14X	19*	28X	31*	18*
0:19	9:17	8:38	9:33	9:10	8:31	9:02	14:21	8:31	8:31
20*	20	25*	22*	27*	22X	22*	21*	27*	32*
8:31	5:19	8:38	8:32	9:02	8:32	8:36	8:35	9:17	8:37
23X	14X	31X	21X	23*	18*	27*	27*	26X	20*
9:09	8:31	8:31	8:31	8:30	8:31	9:34	8:31	8:41	8:31
28X	26X	14X	21	25X	19*	24*	31*	22*	25*
11:09	11:11	8:31	1:10	9:34	8:30	8:36	9:34	8:31	9:02

t :	total number of trials
s :	number of successful trials
r :	probability of success
x :	number of trials up to and including the first success
PS_{avg} :	average number of packets required during a successful trial
PF_{avg} :	average number of packets exchanged before a failure
TS_{avg} :	average time required during a successful trial
TF_{avg} :	average time before a failure
P_D :	total number of packets required for full data transfer
T_D :	total time required for full data transfer
P_{total} :	number of packets (including association set up) required to receive full response
T_{total} :	time elapsed (including association set up) before full response is received

A successful trial indicates that the entire table could be retrieved. Failure indicates that a connection was timed out before complete retrieval. A **deadlock** may occur when one side closes the connection and sends a RESET packet which is dropped. The other side will not detect the closure if it has nothing to send. Deadlocks predominate during the use of linked replies at high rates of packet loss. Therefore if a failure is due to a deadlock, the time for failure is considered to be the time from the start of data transfer to the moment when the agent closes the connection. The following calculations are based on the data in Table 4.13. Note that PS_{avg} was calculated by summing the packets required during a successful trial and then dividing by the total number of successful trials. The other averages were calculated similarly. The values for P_A and T_A have been obtained from Table 4.5. Table 4.16 shows the final results for the high values of q .

$$q = 0.6, \quad t = 50, \quad s = 30$$

$$r = \frac{s}{t} = \frac{30}{50} = 0.60$$

$$x = \frac{1}{r} = 1.67$$

$$PS_{avg} = 17.63, \quad PF_{avg} = 27.40$$

$$TS_{avg} = 1 : 58, \quad TF_{avg} = 8 : 57$$

$$P_D = (x - 1)PF_{avg} + PS_{avg} = 35.99$$

$$T_D = (x - 1)TF_{avg} + TS_{avg} = 7 : 58$$

$$P_{total} = xP_A + P_D = 88.68$$

$$T_{total} = xT_A + T_D = 20 : 30$$

Table 4.16: The P_{total} and T_{total} values for $q = 0.6$ to 0.8 .

q	x	P_A (packets)	T_A (mins:secs)	P_D (packets)	T_D (mins:secs)	P_{total} (packets)	T_{total} (hrs:mins:secs)
0.6	1.67	31.55	7:30	35.99	7:58	88.68	00:20:30
0.7	3.33	44.61	10:11	79.45	24:35	228.00	00:58:30
0.8	10.00	155.70	52:14	231.18	83:20	1788.18	10:05:40

Figure 4.6 graphs the percentage of timeouts incurred by using separate replies, i.e., where every query retrieves one object only, and linked replies, for various rates of packet loss. Ten objects were recovered in each case. Since the number of packets used for delivering the linked replies was very low in this case, a larger number of data transfers completed successfully when linked replies were used. However, when the number of objects recovered increased to 50, the number of timeouts while using linked replies approaches the number of timeouts incurred while using the “single query – single object” approach.

Table 4.17 presents the total number of packets (including association establishment and data transfer) required to retrieve 10 objects by the two mechanisms. At low q values note that linked replies take fewer packets because fewer connections are lost and the data transfer phase itself takes fewer packets to complete. With higher q values this is offset by increasing connection losses and the fact that more packets are required per failed attempt. In addition, because of the “all or nothing” nature of operations involving linked replies, the entire operation has to be repeated, unlike the single reply at a time situation where only the remainder of the transaction need be repeated.

Conclusions

Under conditions of low packet loss, the linked reply approach is superior to the retrieval of each entry separately. This happens because fewer packets are required to transmit all the recovered object classes to the manager, and the manager uses only a single query. While extracting 10 objects separately, approximately 42 packets are required under conditions of no loss, whereas the use of linked replies would require approximately 18 packets (Table 4.17).

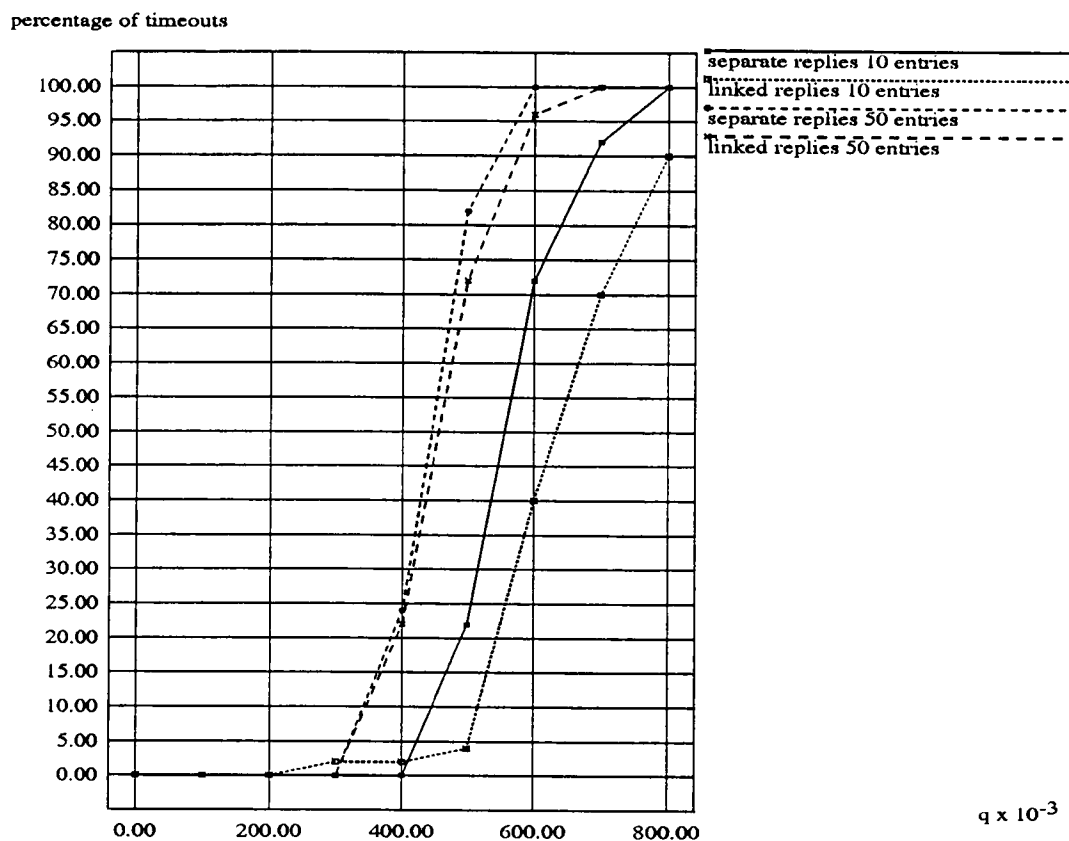


Figure 4.6: The percentage of timeouts vs. the packet drop rate.

Table 4.17: The mean number of packets (including association establishment and data transfer) required by the two modes of data transfer.

q	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
single	41.90	45.74	51.10	59.22	69.74	81.62	135.88	228.51	1251.29
linked	17.72	19.92	22.44	26.50	29.68	35.34	88.68	228.00	1788.88

Fewer packets on the wire translates directly to low operation completion times and fewer chances of dropping the connection under lossy conditions. However, these

benefits depend on the number of objects being recovered and the packet loss rate. If a large number of objects are being recovered, deterioration in service will be noticed at lower rates of packet loss.

However, under conditions of high packet loss, the linked reply strategy has a serious disadvantage. This stems from the fact that TCP does not guarantee the reliable delivery of acknowledgement packets which contain no data. While using linked replies, only the agent sends packets to the manager. **This increases the chances of a timeout at high loss**, for even if a segment reaches the manager, the acknowledgement may be lost, resulting in another burst of retransmissions, and so on. However, while recovering one object at a time, the successfully transmitted response would be followed by an acknowledgement as well as a new query. The new query would also contain the same acknowledgement, and it would continue to be retransmitted till it reaches the agent. therefore there is a better chance of the acknowledgement reaching the agent before a connection timeout. This problem leads to many lost connections at high values of q , especially if the number of objects being recovered is also large.

Deadlocks become more frequent at high q values. This happens when the agent times out and sends a reset packet the manager. This reset packet however is dropped. Since the manager has nothing to send, it will wait infinitely (unless keepalive packets are used).

As discussed before, the "all or nothing" nature of linked replies could lead to extremely large values of P_{total} and T_{total} compared to recovering one object at a time, this being more noticeable at high values of q . Worse still, currently linked replies are the only way of recovering entire tables in CMIP. **There is therefore a need to formulate an alternate way of recovering entire tables using CMIP at high loss rates that will be more akin to the GET-NEXT operation supported by SNMP.**

4.4 Analysis of the Connectionless Transport Protocol

4.4.1 A Model of the Connectionless Mode of Message Exchange

A connectionless network management protocol, i.e., SNMP, uses single datagrams to encode both queries and responses. The concept of linked replies does not exist. The management protocol or application is in sole charge of retransmission when datagrams are dropped. Figure 4.7 illustrates this situation. Dropped datagrams are designated by arrows that do not reach the other side. *The assumption is that the retransmission*

interval is large enough, so that the manager does not retransmit when that query or the associated response is actually in transit. That would result in duplicate responses being received which the manager would then have to discard.

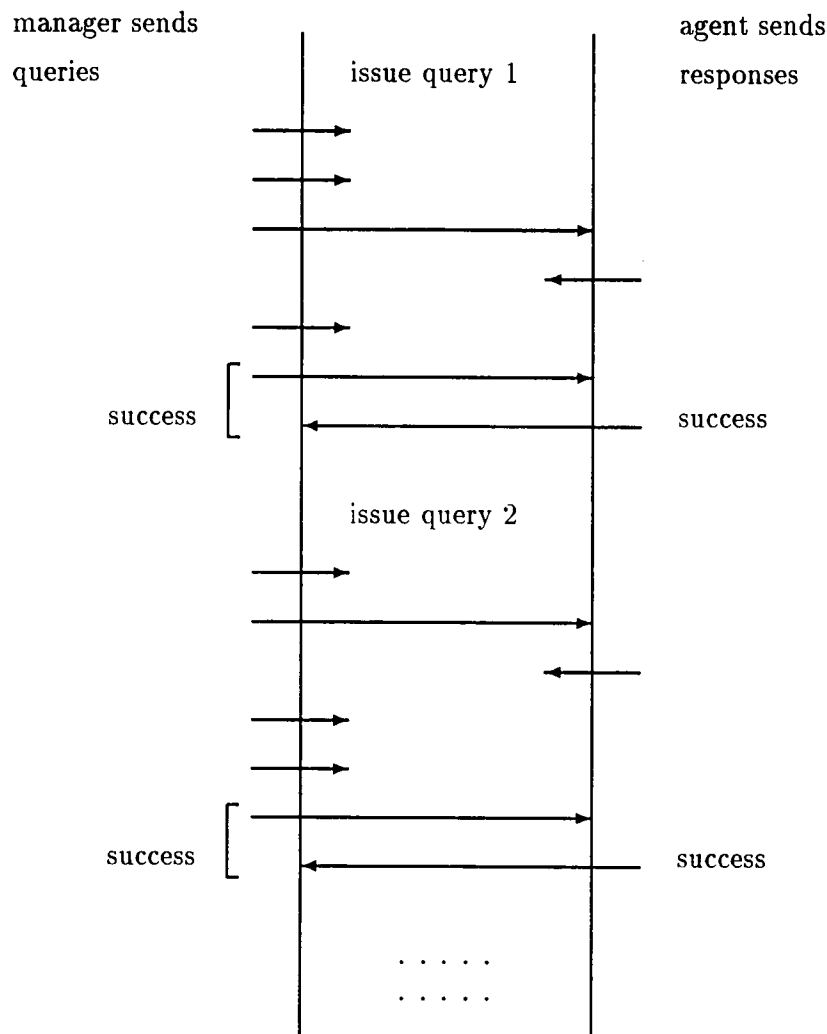


Figure 4.7: The connectionless mode of data transfer.

Assume that the probability of a datagram reaching its destination is p . The manager records a successful trial when its query reaches the agent and the corresponding response is received by it. The probability of both packets reaching their destination is p^2 . Therefore the manager is expected to perform $\frac{1}{p^2}$ transmissions to achieve success, i.e., receive a response. Since the manager transmits one datagram (query) per trial, it will transmit $\frac{1}{p^2}$ datagrams on an average in order to receive a response. Consider

the agent, it records a success when a response reaches the manager, the probability of which is p . Therefore the agent is expected to perform $\frac{1}{p}$ transmissions in order to achieve one successful transfer of the response. Adding the manager and the agent's transmissions, it is seen that $(\frac{1}{p^2} + \frac{1}{p})$ datagrams are expected to be transmitted before a response is received by the manager. For n consecutive queries, a query being transmitted only after the response to the previous query has been received, the expected number of transmissions is

$$P_D = n(\frac{1}{p^2} + \frac{1}{p}) \quad (4.7)$$

If the manager retransmits a query after s seconds, $s(\frac{1}{p^2} - 1)$ seconds will elapse before the response is received. If n queries are used, the time required to complete data transfer is

$$T_D = ns(\frac{1}{p^2} - 1) \quad (4.8)$$

It is also possible to derive an expression for the variance⁸ in the time required to complete the transfer. This will be a simple multiple of the variance in the number of packets transmitted by the manager. For every response received,

$var(\text{datagrams transmitted by manager per response})$

$$= \frac{1 - p^2}{p^4}$$

We know that the sum of the variances equals the variance of the sum for independent events. Since it may be assumed that the transaction required for receiving each response is an independent activity, the variance for receiving n responses becomes

$$var(\text{total datagrams transmitted by manager}) = n(\frac{1 - p^2}{p^4})$$

The standard deviation for the number of datagrams transmitted by the manager can now be written as

$$\sigma = \sqrt{n(\frac{1 - p^2}{p^4})}^{\frac{1}{2}}$$

Since the interval between retransmissions is s seconds, the standard deviation of the time required to complete data transfer is

$$\sigma_T = s\sigma \quad (4.9)$$

⁸Note that the distribution of the number of trials up to and including the first success form a geometric distribution. If a is the probability of success, the mean of such a distribution will be given by $\frac{1}{a}$, the variance being $\frac{1-a}{a^2}$.

Calculating the variance of the total number of packets transmitted by the manager and the agent is a more involved process and been kept outside the scope of this work. The difficulty is due to the fact that the number of transmissions from the manager and the agent are not independent.

Table 4.18: The number of packets and the time required to retrieve 10 objects with SNMP.

q	P_D pred. (packets)	P_D expt. (packets)	$sd(P_D)$ expt. (packets)	T_D pred. mm:ss	T_D expt. mm:ss	$sd(T_D)$ pred. mm:ss	$sd(T_D)$ expt. mm:ss
0.0	20.00	20.00	0.00	constant ⁹	0:01	0.00	0.00
0.1	23.5	23.62	3.19	0:05	0:05	0:03	0:04
0.2	28.12	27.42	3.85	0:11	0:11	0:06	0:05
0.3	34.69	33.64	6.65	0:21	0:20	0:09	0:10
0.4	44.45	44.34	10.72	0:36	0:39	0:14	0:21
0.5	60.00	60.50	14.88	1:00	1:03	0:22	0:22
0.6	87.50	87.24	23.90	1:45	1:49	0:36	0:39
0.7	144.44	140.46	42.00	3:22	3:21	1:07	1:11
0.8	300.00	291.46	83.68	8:00	8:07	2:35	2:37
0.9	1100.00			33:00		10:29	

Experimental Validation of the Model

Table 4.18 provides the predicted (pred.) and experimental (expt.) values of the number of packets and the time required to retrieve 10 objects over a connectionless network management protocol, running over a connectionless transport protocol, (SNMP running over UDP in this case). For each value of q , 50 observations were used with $s = 2$ seconds. The standard deviations are also reported. Note that experiments have not been conducted for $q = 0.0$ since the experimental values would have been the same as the predicted values because no randomness is involved in this case. No experiments have been conducted with $q = 0.9$ either, because of the excessive time required to conduct the experiments. Predictions indicate that 33 minutes will be required for each observation. Fifty observations would therefore require approximately 28 hours.

A comparison of the experimental values with the predicted values indicates that the analytical model tracks actual behavior closely.

⁹This is the constant amount of time required when no datagrams are dropped. It is part of the recorded time for all values of q .

Conclusions

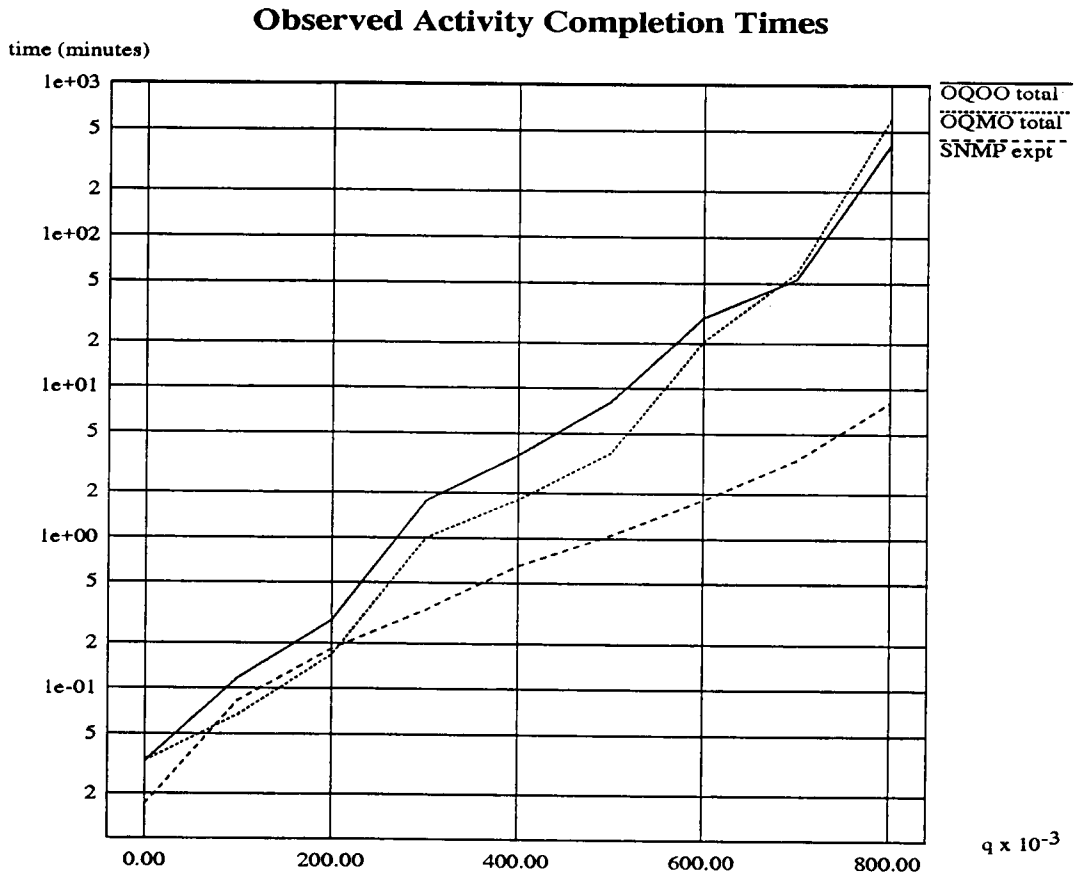


Figure 4.8: Observed activity completion times.

Figure 4.8 shows the average time taken by the various modes of data transfer to complete the retrieval of 10 objects. The times shown include the association establishment times (if any) as well as the time required for actual data transfer. The time required by the connectionless management protocol to complete a transaction is dependent on the time between retransmissions. This time can be configured by the management application. Under normal conditions a relatively high RTO can be used in order to conserve bandwidth. During network fire fighting, a low value should be used to increase the management query's chances of reaching the managed node in the face of heavy packet loss.

It is clear that a connection oriented management protocol will provide acceptable

performance only when the rate of packet loss is low. If a greater number of objects are retrieved, then performance degrades at lower rates of packet loss, especially while using the linked reply strategy, since the greater number of packets required will no longer allow it to "dodge the bullets", i.e., avoid a lossy period during transfer, as effectively as it did when only 10 objects were to be retrieved. Hence, **the advantages of the connection oriented scheme are only realizable at low rates of packet loss.** On the other hand, **the connectionless management strategy provides graceful degradation of service over the entire range of lossiness.**

Chapter 5

Evaluation of Agent Performance

5.1 Objectives and Methodology

The objectives of this chapter are as follows:

1. formulate the basic operations,
2. report performance data,
3. analyze the data in order to evaluate the performance of the different services and mechanisms provided by the agents, and so determine
 - the aspects of CMIP and SNMP implementations that lead to poor performance (question 2c) and
 - the host impacts of the functional groups of CMIP (question 2d).

A set of basic operations is defined according to the guidelines presented in Section 3.1. These operations are the basis of the evaluation process that follows. Their properties are reiterated here for ease of reference. A set of basic operations has the following properties.

1. It is the smallest and simplest set of operations that exercises all the services and mechanisms offered by the agent.
2. It covers all the operations that are commonly performed by the management station.

The performance data consists of the execution times of the basic operations on the agent station. These data are affected by the protocol mechanism itself and by the efficiency or lack thereof of the implementations being used for the tests. It is important to keep this in mind when the measurements are used to judge the services and mechanisms offered by different protocols.

A comparative performance evaluation of protocols cannot always be made in isolation from actual implementations of the protocols [69, 72]. The analyst should ensure that these implementations use the same software tools and hardware platform as far as possible. In the present context, implementations of the CMIP and SNMP protocols are available. They both use the C programming language and the ISODE ASN.1 tools. They also use the ISODE transport layer to interface with either TCP or UDP. In addition, CMIP uses the ISODE application, presentation, and session layers. These layers are not utilized, nor required by SNMP.

While trying to evaluate the services and mechanisms offered by a protocol, the analyst should make every effort to separate out implementation dependent effects from effects that are intrinsic to the service or mechanism being analyzed. This will require alertness, a good knowledge of the implementation, as well as an understanding of the protocols being tested.

The *gprof* profiling facility is used to gather performance data. *Gprof* uses a statistical method to determine the time taken by each function in the code [44]. The program counter is sampled periodically. *Gprof* calculates final times by determining how many times the program counter was found within the range of addresses covered by the function. If the function does not run long enough, *gprof* might miscalculate its execution time, as it will not be able to gather a proper sample. This consideration is used to determine how many management operations are to make up each span of measurement. *Gprof* samples the program counter approximately every 10 ms. Therefore, each function must run for at least 100 ms, which is an order of magnitude higher than the sampling interval. Since most functions of interest took more than 0.05 ms, the number of management operations per measurement was fixed at $(100 \text{ ms}) / (0.05 \text{ ms})$, i.e., 2000 operations. The agent is terminated after the measurement session, as *gprof* generates the files of performance data only when the program ends. Some agent daemons may need special instrumentation to achieve this as not all agents may be started or terminated by the manager. Each set of 2000 operations is performed 10 times. The rationale behind using this sample size is presented in the next section. The mean of the 10 readings is the most likely time taken by a function. In this context, the excellent discussion on choosing the mean, median, or mode for summarizing

performance data provided in [39, pp. 183 – 185] is instructive. The mean is the most appropriate measure in our situation, as there is no distinct skew in the data.

It is important to have a good understanding of one's measurement tool. Users of *gprof*, for instance, have to note that sometimes a function can exit conditionally, without doing its designated job. The following is an example:

```
int example_function ()
{
    if (condition1)
        exit(0);
    else {
        ...
        perform
        designated
        job
        ...
    }
}
```

In such cases *Gprof* may report values that are lower than the correct value for such functions. If there are ten premature exits from the function and ten full runs, the total time spent in `example_function()` will be divided by 20, rather than 10. This will lower the reported average running time of the function drastically. Graham, *et al.* [43] have this to say about *gprof*, “We make the simplifying assumption that all calls to a specific routine require the same amount of time to execute. This assumption may disguise that some calls (or worse, some call sites) always invoke the routine such that its execution time is faster (or slower) than the average time for that routine.”

5.1.1 Determining the Sample Size

The trust that can be placed in conclusions drawn from a set of measured data depends upon the size of the data set, i.e., the sample size. A larger sample size increases the confidence level. However, it also requires more time and resources to gather additional data. It is therefore desirable to use the smallest sample size that provides an acceptable level of confidence.

In the present situation, the execution times of a large number of functions are to be analyzed. Since the functions have widely varying runtimes, *gprof* will report the times

with different degrees of variance, since it can measure longer running functions more accurately than functions that take less time to execute. Therefore, if the accuracy and the confidence level required were fixed, different sample sizes would be required for each function in order to achieve that accuracy. However, changing the sample size for each function is not practical because then data collection cannot easily be automated. Besides, in some situations the number of samples required may be prohibitively large since the sample size varies with the square of the desired accuracy. Fortunately, it is not necessary for all functions to be measured with the same accuracy. Furthermore, run times that are relatively insignificant need not be measured accurately, since only the trend is of interest here. This will permit the use of a fixed sample size. Fixing the number of repetitions of an operation at 2000, brought visual consistency to the data. Each set of 2000 operations was repeated 10 times. It was observed that a sample size of 10 yields confidence intervals between 1% and 10% for almost every function at a 95% level of confidence.

The following formula has been utilized for all calculations [39, p. 217].

$$n = \left(\frac{100zs}{r\bar{x}} \right)^2$$

$$\Rightarrow r = \frac{100zs}{\sqrt{n\bar{x}}} \quad (5.1)$$

where

$z = 1.96$ for a confidence level of 95%

s is the standard deviation of the observed data

$\bar{x}$ is the sample mean

r is the desired accuracy expressed as a percentage, and

n is the sample size

Equation 5.1 was applied to the observed data to determine the confidence intervals of the sample means at a 95% confidence level. The results are presented in Table 5.1. Functions that have a runtime less than 0.10 msec. have the widest confidence interval. In the observed data, only CMIP's filtering function exhibited such low execution times. The wider confidence interval in such cases is not a drawback since it will not affect the conclusions drawn from the experiments.

The confidence intervals shown in Table 5.1 are acceptable, since the goal of this study is the identification of protocol dependent trends. Extreme precision is not necessary while evaluating the protocol's services and mechanisms (as opposed to evaluating

Table 5.1: Confidence intervals of the sample means at a 95% confidence level.

Range of $\bar{x}$ (msec)	Confidence interval
> 10.00	2%
1.00 – 10.00	5%
0.10 – 0.99	10%
< 0.10	15%

the implementation itself) because the measured runtimes have already been affected by implementation and system dependent factors in an unknown manner.

The confidence interval widens as the sample mean decreases. This happens because the total runtime of such a function is also low (even after repeating it 2000 times). The measurements made by *gprof* therefore exhibit a larger variance. From Equation 5.1 it is immediately apparent that a higher variance in the sample leads to a wider confidence interval, all other parameters remaining constant.

5.1.2 Selecting the Test Object Classes

Since the MIB used in the current work is relatively small — the OIM MIB contains 24 object classes, of which only 16 are significant for evaluation purposes — it was possible to utilize the majority of them while conducting the measurements. However, there will be cases when the number of object classes will be large, perhaps fifty or more. In such a situation, it would be impractical to utilize all of them in the tests, because then the total time required would be prohibitive. The time taken to run the whole suite of tests would then clearly become unacceptable. It is therefore necessary to select a representative set of object classes which may be used to formulate the tests.

From the experience gained during the current work, certain empirical rules can be formed to guide the selection of suitable object classes.

1. About 15 object classes should be chosen for the tests. They must be representative of all the object classes.
2. Object classes which are most likely to be used during the management of real networks are to be chosen.
3. The object classes should be a mix of the following:

- (a) Table entry objects (both dense tables like `atEntry`, where table entries are indexed sequentially, and sparse tables like `udpEntry`, where table entries are indexed by numbers that are not in sequence — this affects the efficiency of the SNMP GET-NEXT service).
 - (b) Non table entry objects like the `icmp` object class.
 - (c) Object classes that require extensive kernel access, like the `tcp` and `udpEntry` object classes.
 - (d) Object classes that require no kernel access after agent initialization (`interfaces`) or little kernel access at each operation on it(`ip`).
 - (e) Object classes with multiple instances and object classes with a single instance. The number of instances often determines the retrieval time.
4. For tests that utilize scoping (or accessing different object classes in some fashion), a maximum scoping level of 3 should be sufficient to establish the credentials of the protocol/implementation under test. Care should be taken to see that the scope covers a non-trivial number of object classes and instances of those object classes. The depth of scoping may be increased or decreased depending on the number of object classes and object instances that fall within the scope. The aim is to avoid too wide or too narrow a scoping range.
 5. Classes which work in a highly implementation dependent fashion should not be selected. Consider the class `system` for example, some agents may choose to write through to a file, thereby increasing the execution time. Other agents may just update a cache, without a write through, thereby avoiding time consuming file I/O.
 6. Objects that may be created or deleted, and those that generate events are to be included.

It is best to confine the experiment to the same 15 or so object classes for the entire suite of tests, as this makes the process of interpreting the results more manageable, since one can concentrate on the functioning of fewer object classes.

It may not be possible to satisfy all these constraints while testing real systems. Liberties may therefore be taken in the object class selection process, but the goal of this process should be kept in mind throughout — *these selected object classes should yield results in an acceptable amount of time, that are indicative of the protocol's performance over all the allowable object classes*. There should be a balance between the time invested and the amount of useful performance data acquired.

5.1.3 Formulating the Basic Operations

The following subsections are devoted to formulating the basic operations defined in Chapter 3. These operations will cover the GET, SET, CREATE, DELETE, and EVENT-REPORT services.

Formulating Tests for the GET Operation

For both CMIP and SNMP, the GET operation may be specified in a variety of formats, each of which performs differently. Figure 5.1 shows the types of GET operations which were deemed necessary for evaluating the performance of a network management protocol. Other possible formats have been examined and discarded as they contributed little or nothing to the understanding of the protocols. They will not be included in this report. The contribution of each of these basic operations to the performance evaluation process will now be considered.

1. Retrieval of a single object instance without scoping or filtering. This is the operation that a CMIP style management protocol performs best. Measurements in this category will tell how economically the protocol retrieves all attributes from a given object instance. Since this operation will be used often by the manager, it provides a good metric for the evaluation of different management protocols.
2. Retrieval of single object instances each from two different object classes using scoping and filtering where applicable. Protocols like SNMP do not allow explicit scoping and filtering, hence the query set has to be constructed accordingly for such protocols.

This will be compared to the results in category 1 above, in order to determine the additional expense associated with scoping. As noted before, a scoping operation will always run into unpredictable delays due to the extra object instances that might become part of the range of the scope.

However, the appeal of scoping is in the ease with which it enables the manager to specify relatively complex operations. It also reduces the amount of network traffic necessary to complete the operation, as the manager uses one query, rather than several. Scoping is most useful when it is used on objects in the same class, table entries for instance. In such a situation, there are no long delays as the scope does not encompass many undesired object instances.

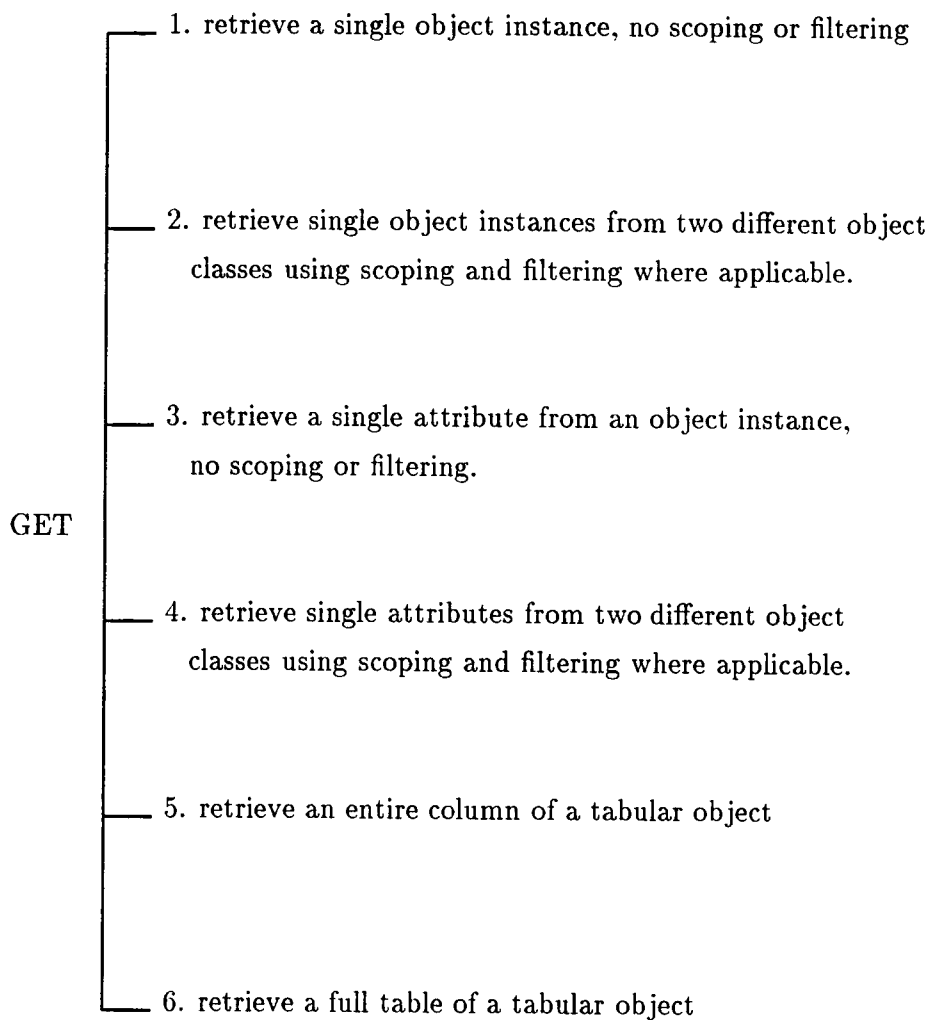


Figure 5.1: The GET operation formats used in performance evaluation.

Note: In CMIP, the attributes present in the filter used with the scope may be looked up in a table to see what object classes they correspond to. Then the agent only applies the filter to those object classes while it traverses the MIB tree that is covered by the scope. This will save it from examining unwanted object classes.

3. Retrieval of a single attribute from a single object class without scoping and filtering. This is analogous to 1, except that it is at the attribute level. Tests 3 and 4 have been designed specially for SNMP style protocols which work at the attribute level rather than at the object class level.
4. Retrieval of single attributes each from two different object instances that belong to different object classes. These measurements will be used in conjunction with test 3 to determine the additional overhead required to process multiple attributes with a single query, especially in SNMP style protocols.
5. Retrieval of a full column of a tabular object. The mechanism for doing this is protocol dependent. Since this is a key management operation, it provides valuable information about a protocol's table traversal mechanism.
6. Retrieval of the full table of a tabular object. This is an often used management operation.

Formulating Tests for the SET Operation

Theoretically, all the variety that was observed in the GET operation is also permitted with the SET operation. However there is no need to measure them all due to the following reasons:

1. The SET operation is permitted by only a small subset of the managed objects. This limits the filtering and scoping that may be used with this operation.
2. It can be assumed that the mechanism used to access multiple object instances with a single SET operation is identical to that used during a GET operation. Since the delays associated with multiple object selection have already been examined in the previous section, there is nothing to be gained by exercising the scope and filter facilities again.

The SET operations are most readily divided into the following groups:

- Setting a single attribute in a scalar object.

- Setting a single instance of a vector object.

In this context, a scalar is an object class whose writable attributes are independent of each other, i.e., `ip` and `ifEntry`. A vector is an object whose writable attributes are interdependent, i.e., `ipRouteEntry` and `ipNetToMediaEntry`.

Formulating Tests for the CREATE and DELETE Operations

Running tests in this category is complicated by the fact that one cannot create 2000 instances of an object class continuously because few object classes in MIBs allow 2000 possible instances on real systems. The same complication arises with the DELETE operation, as it is not possible to delete 2000 instances because that many instances may not exist to begin with. In order to circumvent this hindrance, the code is augmented so that after every CREATE operation, an `ioctl` system call (on Unix systems) is used to delete it, so that the next CREATE operation can create it again. Similarly, after every DELETE operation, an `ioctl` is performed to recreate that instance, so that it can be deleted again with the next DELETE command. The time necessary for the additional `ioctl` is subtracted to obtain the actual run times for the CREATE and DELETE operations.

The CREATE operation does not allow any filtering or scoping. The DELETE operation allows both. However, the scoping and filtering will be ignored because the reasons provided for the SET operation are also applicable here.

Formulating Tests for the EVENT-REPORT Operation

In order to generate performance data, the agent must be terminated after 2000 event-reports have been delivered to the manager. In agents that are capable of generating more than one event, a mix of events should be used for the measurement sessions. In this study however, each agent was configured to generate the same event repeatedly.

5.1.4 Controlling the Experimental Environment

In order to generate performance results that are fair and reproducible, the experimental environment must be controlled. The factors mentioned here need to be controlled, or monitored if control is not possible. These factors should be reported along with the results.

Factors External to the Manager and Agent

1. Hardware and operating system

The hardware and operating system on which the agent is executing has a great influence on the performance data. The effect of hardware speed is obvious. The operating system influences the running time because of the way it chooses to store the managed objects, especially the table entries. A good example is the *tcpCurrEstab* attribute, which is the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT. Unix does not keep a record of this number, hence the agent must obtain it by traversing all the TCP entries in the kernel and counting the entries in the desired states. The large number of kernel fetches increases the retrieval time for this attribute greatly. This would not be the case had the agent been running on an operating system that kept a record of this counter.

2. System load

The load on the machine on which the agent is running has an impact on the times reported by *gprof*. As the load on the machine rises, the agent will accumulate more running time due to the extra time for the page faults and slow downs caused by extra context switches [44]. To combat this, the following steps were taken.

- (a) Logins were disabled after checking that no one was logged on already. People could still login as super user, but that is rare in the experimental environment. The measurements were done as batch jobs via the Unix *at* utility. No user logins were active on the agent machine during the time when the measurements were conducted.
- (b) The manager was run on a different machine, as it would affect the agent's running time had it been executing on the same machine as the agent. The manager does not have to run in a controlled environment. It is assumed that the standard daemons present on the agent machine (*inetd*, *lpd*, etc.) did not undergo significant activity while the experiments were in progress.

3. The MIB

The MIB should be identical as far as possible for all protocols being evaluated. There may be slight differences in the MIBs since different protocols may use different SMIs. For instance, the OIM MIB specifies a separate distinguished attribute for each object class (usually the name of the class itself with the suffix

"Id"). However the Internet MIB does not provide such an attribute. Because of this, it will often be observed that while recovering all the attributes of an object class, CMIP will recover one more attribute than SNMP. The object classes supported in the current MIBs include the following: **system**, **interfaces**, **at**, **ip**, **icmp**, **tcp**, **udp**, **egp**, and **snmp**. The number of objects in the MIB determine the time and memory required to construct the run-time MIB tree. The size of this tree then determines the time taken by the agent to traverse it. Although the **egp** and **snmp** groups are part of the MIB, the agents themselves do not permit management operations on these two groups since the experimental platforms do not support the Exterior Gateway Protocol which is essentially a special purpose routing protocol. The **snmp** group is not supported by the agent because if the CMIP agent were to support it, there would be a need to have a concurrently running SNMP agent export the attributes in the **snmp** group to it. This is an additional complexity that does not contribute to the main goal of this study. Hence, it has been avoided.

4. Number of entries in the managed tabular objects

The following managed objects are table entries [29]:

atEntry These are the Address Resolution Protocol (ARP) table entries.

ipNetToMediaEntry Same as **atEntry**.

ifEntry Entries describing each interface at the physical layer.

ipAddrEntry Entries describing the addressing information related to this node's IP addresses.

ipRouteEntry Entries describing each route presently known to this node.

tcpConnEntry Entries describing each TCP connection that this node is a member of.

udpEntry Entries describing the local IP address and port number for the UDP listeners on this node.

The agent must search through these entries to locate the instance requested by the manager. This can take considerable time in spite of the relatively few entries that populate these tables. This is because each entry requires a kernel read for retrieval (this is true for Unix systems). New entries added to a table will increase the retrieval time for instances in that class. On the other hand, deletion of entries from the table will not only lower retrieval times, but may cause

Table 5.2: The number of instances for the tabular objects.

Object Class	Number of Instances	
	CMIP	SNMP
ifEntry	2	2
ipAddrEntry	2	2
ipRouteEntry	3	3
atEntry	12	12
ipNetToMediaEntry	12	12
tcpConnEntry	23	23
udpEntry	41	42

a query to fail altogether, if an attempt is made to retrieve an object that has been deleted during the course of the measurement. It is important, therefore, that the number of table entries be kept constant or closely monitored during the measurements. The `ifEntry`, `ipAddrEntry`, and `ipRouteEntry` tables usually do not change rapidly over time, hence they require no special control. However, the number of entries in each of these tables should be noted and presented along with the performance data.

The `atEntry` (`ipNetToMediaEntry`), `tcpConnEntry`, and `udpEntry` tables pose special problems as they can change dynamically and quickly over time. The instances of `atEntry` (`ipNetToMediaEntry`) were marked `permanent` in the ARP table. This does not prevent the addition of new entries, it only prevents the deletion of the entries marked `permanent`. The addition of entries to this and the `tcpConnEntry` and `udpEntry` tables can only be prevented by forestalling network communications with other machines (or itself) during the measurements. The steps mentioned in item 2, above, will achieve this end to a limited extent. As before, the number of entries in each of these tables is noted and presented along with the performance data.

Table 5.2 shows the number of instances of these managed objects that were present during the performance measurements. Entries in the address translation table will be counted twice, once for each of the two network interfaces present on the current system.

Factors Internal to the Manager and Agent

1. The section of code whose running time is measured.

One has to decide which functions or sections of code are to be measured. It is easier to measure the times taken by the different functional components of the agent (i.e., encoding, decoding, etc.), if those components are coded as distinct functions, instead of being distributed throughout the code. In these experiments, the time required by the agent to perform a management operation is the time from when the BER (Basic Encoding Rules) encoded query is decoded to the time when the agent's response is encoded into a BER encoded PDU. The encoding and decoding times are also included in the measurement. For SNMP, operating in the Internet protocol suite, this is the time between the moment after the query is read from the socket to the moment before writing the result to the socket. The encoding and decoding times referred to here are the times needed to convert from C structures to the BER encoded data stream and vice versa, using the automatically generated functions provided by the ISODE.

2. The amount of logging done by the agent.

It is best to disable logging altogether. If logging is done, the extent of logging should be similar during all measurements. The former approach has been taken here.

3. The number of attributes in each object class.

Some implementations may not implement some attributes for an object class, report a constant (0 or -1) when the actual data is not available, or compute the value of an attribute incorrectly (one SNMP implementation reports *tcpCurr-Estab* as the total number of TCP connections, rather than only the number of TCP connections in the ESTABLISHED or CLOSE-WAIT state). These factors are to be kept in mind while interpreting the performance results. In the current work, unimplemented attributes were filled in and caused to return a constant (0 or -1) on being queried. This ensures that all protocols process the same number of attributes.

4. The instances used in the workload.

The object instances chosen for conducting a test have a marked effect on object classes like *tcpConnEntry* and *udpEntry*. It takes longer to retrieve an instance that lies at the end of the list, hence the workload should use a random (or pseudo random) cross section of the available instances.

5. The manager.

The manager does not have to be run in a controlled environment, as it does not affect the agent timings reported by *gprof*. This is because the agent and the manager run on different machines, and the time spent by the agent to listen for queries is not part of the measurements.

5.2 The Performance Data and its Analysis

This section presents the performance data related to each basic operation. In each case, a column graph is used to compare the performance of the two protocols being evaluated. These columns are further demarcated to indicate the contribution of different protocol mechanisms to the total time spent in the agent. The total time taken by the *entire* protocol stack has also been marked above the columns. **The data is further analyzed to determine why a protocol performs as it does.**

In any experiment, the accuracy of the measured data is always an issue. The clock on the Sun 4 used in the current study has a granularity of 10 ms. Since each measurement is conducted with 2000 operations, the data for each operation should be accurate up to two places after the decimal. However, the data will also have a margin of error due to the limited number of observations that are used to approximate the population mean of the agent's execution time. The range of these errors is, however, quantifiable as shown in Section 5.1.1. No significance is attached to differences in two data points if the confidence interval of one overlaps the confidence interval of the other.

It should also be noted that for each metric, one has to study the entire output file produced by *gprof* in order to understand the behavior of the agent. This understanding is necessary to explain the causes that lie behind the observed data. Studying the data for a few functions only may produce an incorrect impression about what is actually occurring.

In the following graphs, a section of the time has been marked as **remaining time**, as it could not be conveniently separated out into major components. For CMIP, this represents the time required to convert between the *posy* (an ISODE utility for encoding/decoding between ASN.1 and C structures) generated data structures and a native form that is usable by the agent, finding the object in the MIB tree, and freeing all data structures after a response has been sent. The time required to encode and free will be proportional to the number of attributes being recovered. SNMP must map each managed object identifier in the query to a corresponding MIB entry that

contains its syntax. The remaining time covers this, and therefore varies clearly with the number of attributes being retrieved. While recovering table entries, the time to sort the table entries in lexicographic order is also included under the remaining time heading.

5.2.1 Retrieval Time for a Single Instance of an Object Class

This is the time necessary to recover all the attributes in a single object class. Figure 5.2 gives a graphical representation of these metrics for the CMIP and SNMP protocols. *This is also the time required to retrieve a full row in tabular objects.*

Interpreting the data. The time spent in the CMIP protocol engine itself is lower than that spent in the SNMP protocol engine. However the higher overhead of the CMIP stack causes CMIP to require greater total time in a number of cases. In both cases the encoding time varies with the number of attributes present in each object class. In SNMP, the decoding time is also proportional to this. The section marked 'remaining time' is greater for SNMP as each attribute has to be processed separately. This increase is especially marked for icmp (26 attributes) and ifEntry (22 attributes). Kernel fetch times (includes seeks and reads) are large for tcp, tcpConnEntry, and udpEntry as the latter two classes had a large number of instances (23 and 41-42 respectively). SNMP requires greater kernel fetch time for all classes with multiple instances as the nature of the SNMP protocol makes it more convenient to retrieve all instances and sort them in response to a query. In contrast, the CMIP protocol does better, as only half the table has to be traversed, on an average, in order to recover a given object instance.

If an agent takes more time to retrieve kernel data, it should not be considered a serious protocol shortcoming, since this time is dependent on the hardware and operating system on which the agent runs. Objects requiring long kernel retrieval times on a Unix platform would do much better had the data been more readily available to the agent.

5.2.2 Retrieval Time for an Instance Each from Two Different Object Classes with Scoping and Filtering

This time depends on the depth and width of scoping and the number of object instances encompassed by the scope. A maximum scoping depth of 3 was found to be sufficient to assess the efficacy of the scoping operation. This is also the maximum possible depth of scoping in the OIM MIB. These figures should be studied in conjunction with the

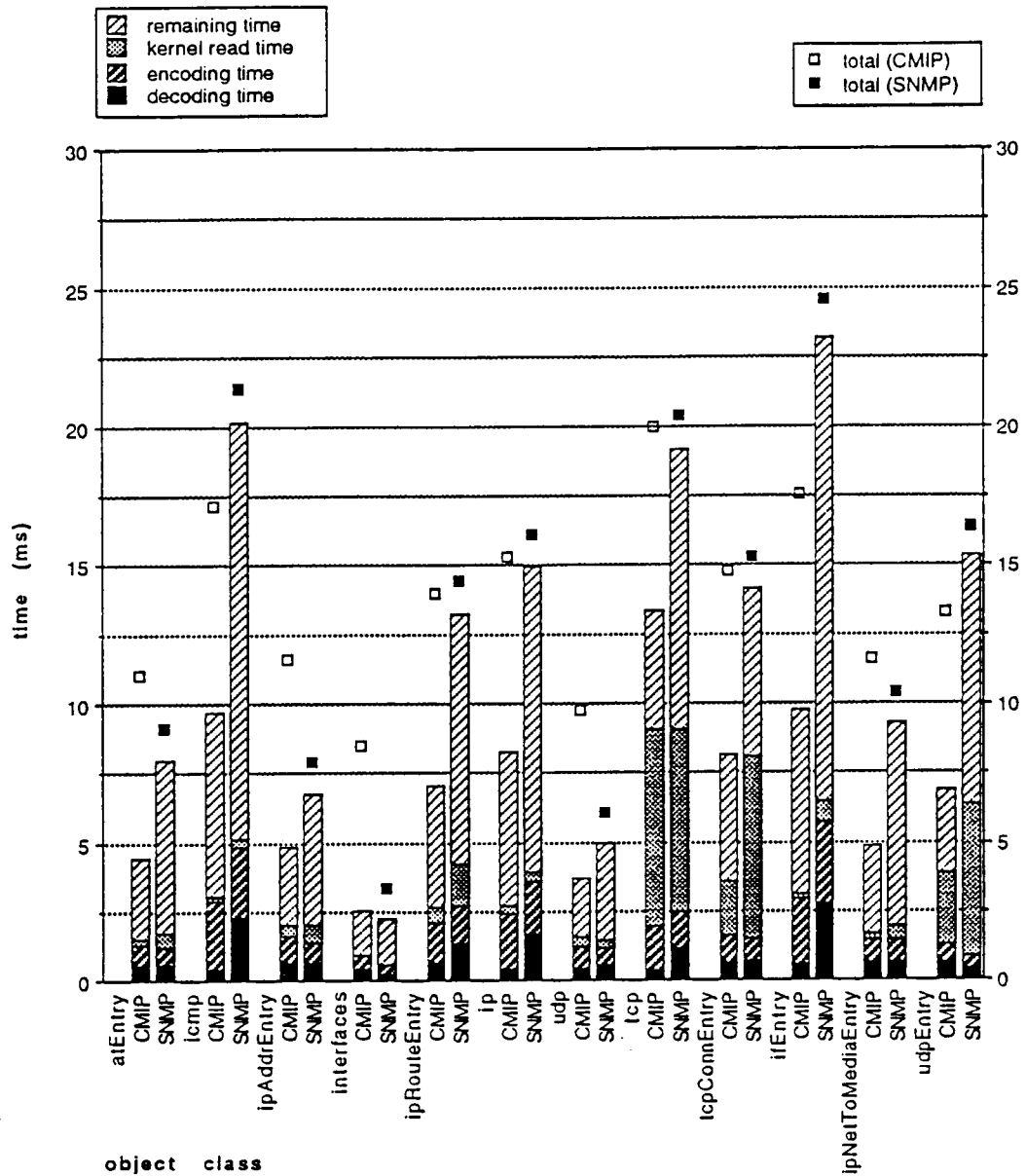


Figure 5.2: CMIP and SNMP data for basic operation 1.

results of the first metric. For instance, if the time taken to recover an instance each from the object classes `udp` and `udpEntry` using scoping is $T_{scoping}$, and that for recovering these instances individually are T_{udp} and $T_{udpEntry}$ respectively (from the first metric), then the expression $T_{scoping} - (T_{udp} + T_{udpEntry})$ should ideally have a value close to 0. A large positive value indicates that a large time period has been spent in filtering out the unwanted instances that were part of the scoped subtree. Although the preceding discussion is more germane to CMIP-style protocols, this is also relevant to SNMP-style protocols and any relative evaluation of the two. Some cases or implementations might yield a value substantially *less* than 0 under special circumstances. The value just discussed is called the **scoping overhead**. The scoping overhead for any group of objects is defined as the time necessary to retrieve those objects using scoping minus the sum of the times necessary to retrieve them individually. Figure 5.3 shows the retrieval times for the CMIP and SNMP protocols. The times necessary for filtering are shown alongside the CMIP data.

Interpreting the data. A scoping overhead somewhat higher or lower than 0 is normal. If this value is much higher than 0, one should study the reasons closely, and check if such a large scoping time is warranted by the extent of scoping used. From the CMIP data it is immediately apparent the more extensive scoping (the category `system3`) takes more time than less extensive scoping (the category `tcp2`). As a matter of fact, in CMIP, scoping across object classes will always have a positive overhead. Such scoping should therefore be avoided unless the underlying network is lossy and one wishes to involve as few PDUs as possible in the transaction. Scoping and filtering is best used to select particular instances from the same object class. Note that the savings in encoding achieved by SNMP, as mentioned later, is absent for CMIP as a separate linked reply PDU is formulated for each instance retrieved. Also note that when a filter is used, the CMIP agent will traverse all instances of a table instead of stopping after finding the required entry. Though this is not dictated by the protocol, implementations will find it convenient to do so since theoretically more than one instance can match a given filter. This necessitates the examination of all available instances. For SNMP, the scoping overhead is expected to be a little lower than 0 as there is no overhead for accessing multiple objects with a single GET. The encoding/decoding times will be marginally less than when two object classes are retrieved with separate GETs. In addition, the query/result will be transmitted via a single PDU, as opposed to the two PDUs that are necessary with the use of separate GET operations. This will decrease the PDU processing time. The overhead for both protocols will decrease as there will be fewer passes through the underlying protocol stack. In some situations

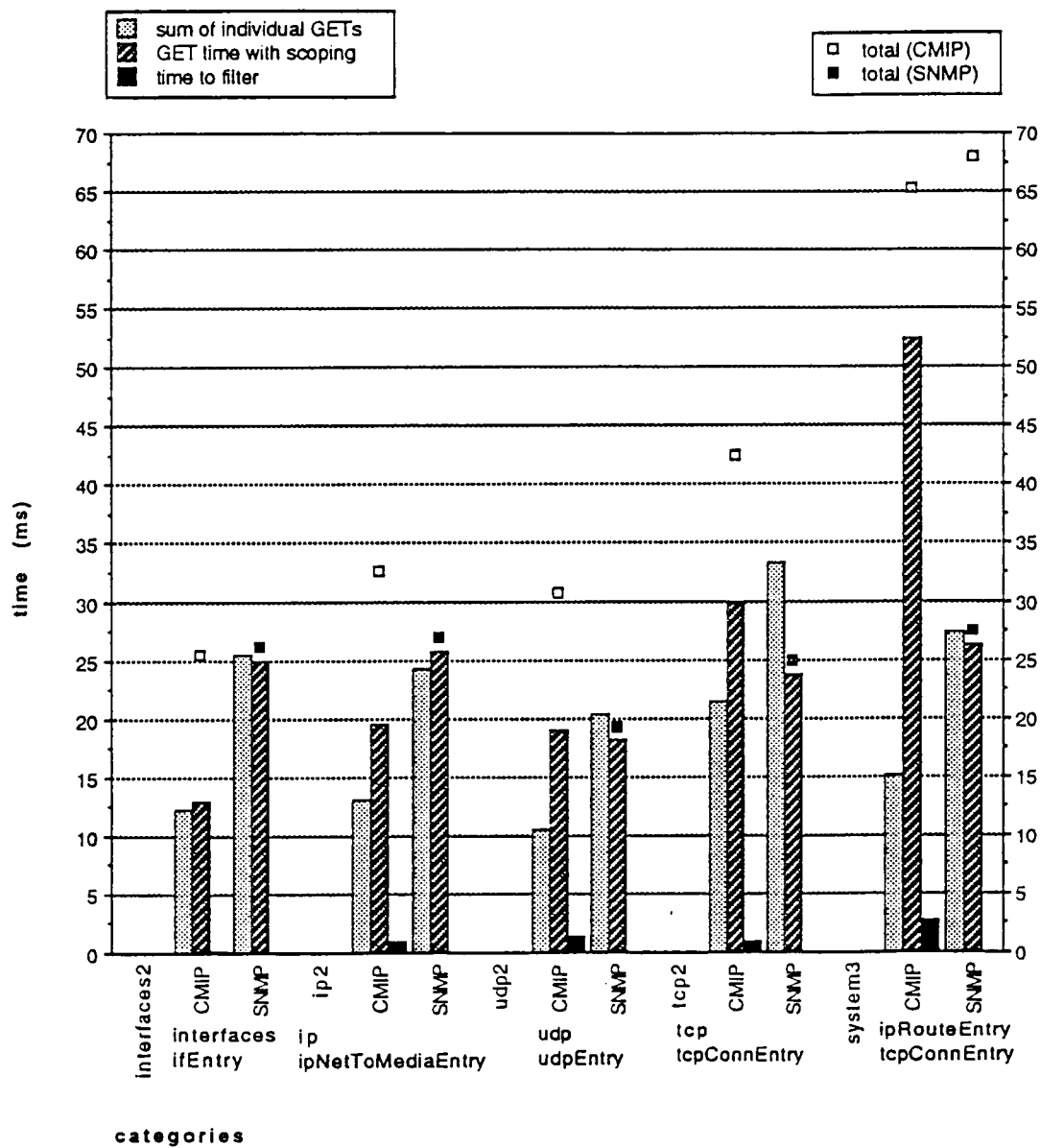


Figure 5.3: CMIP and SNMP data for basic operation 2.

(SNMP category `tcp2`), the scoping overhead turns out to be appreciably less than 0, because while retrieving the attributes from the object class `tcp`, all the instances of `tcpConnEntry` were also recovered in order to obtain the value for the `tcpCurrEstab` attribute. The CMIP implementation does not use this technique, though it could have done so.

5.2.3 Retrieval Time for a Single Attribute

This is very similar to the first metric, except that it is at the attribute level. Figure 5.4 gives a graphical representation of this metric for the CMIP and SNMP protocols.

Interpreting the data. From the data, one may draw conclusions along three lines.

1. *Getting a single attribute vs. getting all the attributes of an object class with SNMP.* Please consider the SNMP graphs in Figures 5.2 and 5.4. The protocol itself leads one to expect that SNMP style protocols can retrieve a single attribute more efficiently than it can multiple attributes. This is because in SNMP the attributes are independent of each other, in theory. However this is not the case on many operating platforms, including the Unix platform, where retrieving a single attribute also involves retrieving related attributes that reside in the same data structure. It has been observed that the decoding and encoding times vary with the number of attributes being recovered. Hence there are no major savings in getting a single attribute over getting all the attributes in the object class if the attributes are stored in a single structure, and if the number of attributes are not many (i.e., `atEntry`, `interfaces`, `udp`, etc.). The saving is significant for the object class `tcp`, as the attribute `tcpCurrEstab` resides in an independent data structure. It has also been observed that the savings are large for object classes like `icmp` (26 attributes), `ipRouteEntry` (24 attributes), `ip` (19 attributes), and `ifEntry` (22 attributes).
2. *Getting a single attribute vs. getting all the attributes of an object class with CMIP.* Please consider the CMIP graphs in Figures 5.2 and 5.4. In theory, the same observations apply here. However, the implementation under test retrieves all the attribute values first and sends out only the ones that have been requested. The saving that was previously seen in the case of the object class `tcp` therefore does not apply here. In fact, a slight overhead will be incurred for selecting the attributes that have actually been asked for by the manager. The only savings are in encoding the result. This saving is substantial in object classes with a

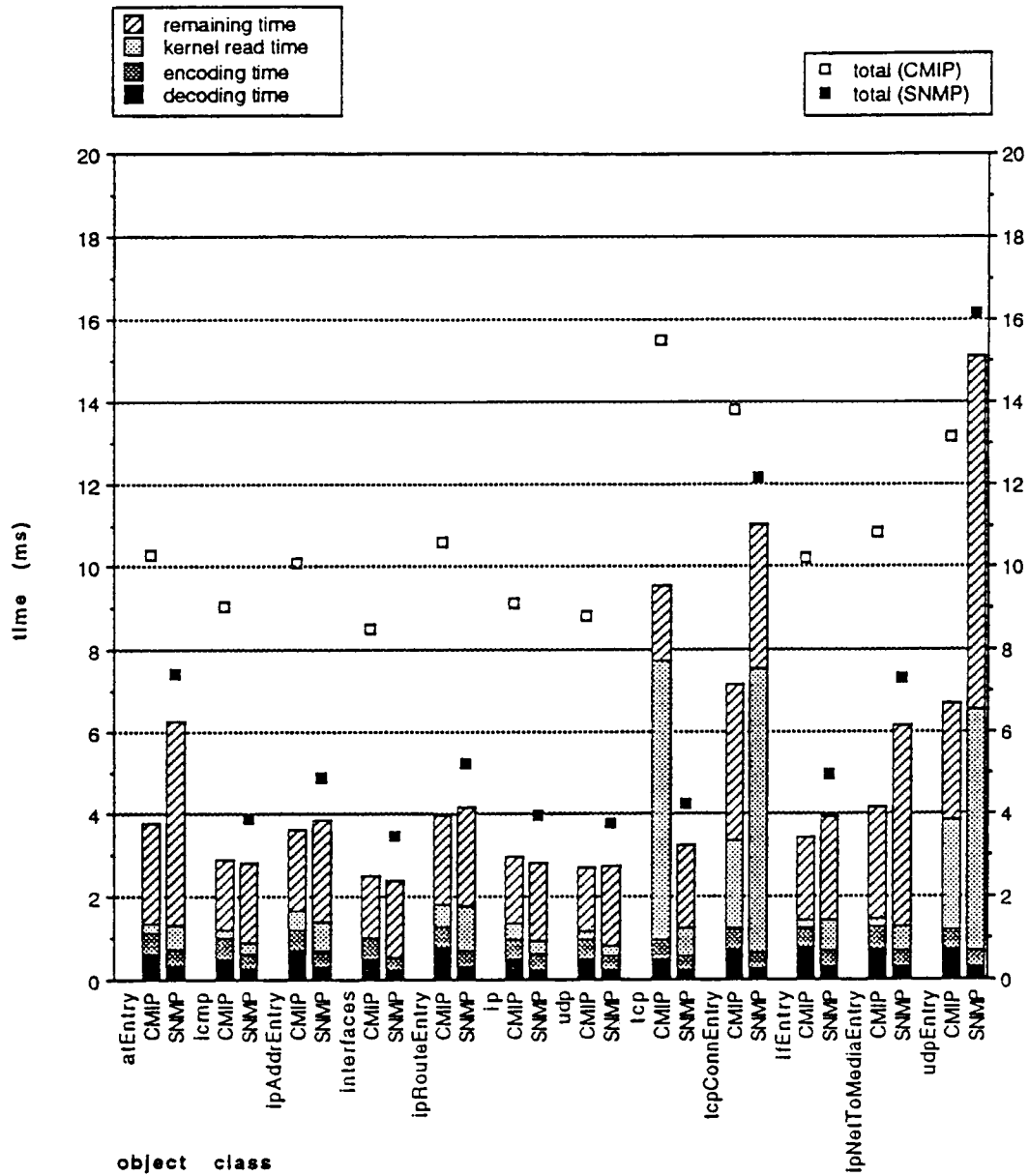


Figure 5.4: CMIP and SNMP data for basic operation 3.

large number of attributes. The decoding time is related to the depth of the requested managed object in the MIB tree, as the deeper its location, the longer is its distinguished name that the agent must decode. This increase should also be experienced while encoding the result, however, the current implementation uses only the relative distinguished name of the target object in the response PDU.

3. *Getting a single attribute with CMIP vs. getting a single attribute with SNMP.* Please consider the two graphs in Figure 5.4. Although it is expected that a CMIP style protocol will take longer, there is nothing in the CMIP protocol itself that forbids an implementation to retrieve only the attributes which are required by the manager. This would of course entail an increase in code size. In the current CMIP implementation however, all attributes are first retrieved. Then the selected ones are transmitted. Although this is not much of an overhead for many object classes, it is quite crippling for classes like `tcp`, where certain attributes (like `tcpCurrEstab`) require extensive additional processing. Retrieving any attribute from the `tcpConnEntry` or `udpEntry` table degrades SNMP performance, since the entire table has to be retrieved.

5.2.4 Retrieval Time for an Attribute Each from Two Different Object Classes with Scoping and Filtering

This metric is being specified mainly for attribute-oriented protocols like SNMP, but it is applicable to CMIP-style protocols also. The goal of this metric is the same as that of metric 2, except that this is at the attribute level. As before, this time depends on the depth and width of scoping and the number of object instances encompassed by the scope (for CMIP). This figure should be studied in conjunction with the results of the third metric. For instance, if the time taken to recover an attribute each from object classes `udp` and `udpEntry` using scoping is $T_{scoping}$, and that for recovering these attributes individually are T_{udp} and $T_{udpEntry}$ respectively (from the third metric), then the scoping overhead $T_{scoping} - (T_{udp} + T_{udpEntry})$ should ideally be close to 0. A large positive value indicates that a large time period has been spent in filtering out the unwanted instances that were part of the scoped subtree. Figure 5.5 gives a graphical representation of these metrics for the CMIP and SNMP protocols.

Interpreting the data. A scoping overhead somewhat higher or lower than 0 is normal. Values much greater than, or appreciably less than 0 should be studied in order to obtain the reason behind such behavior. One should then decide whether such

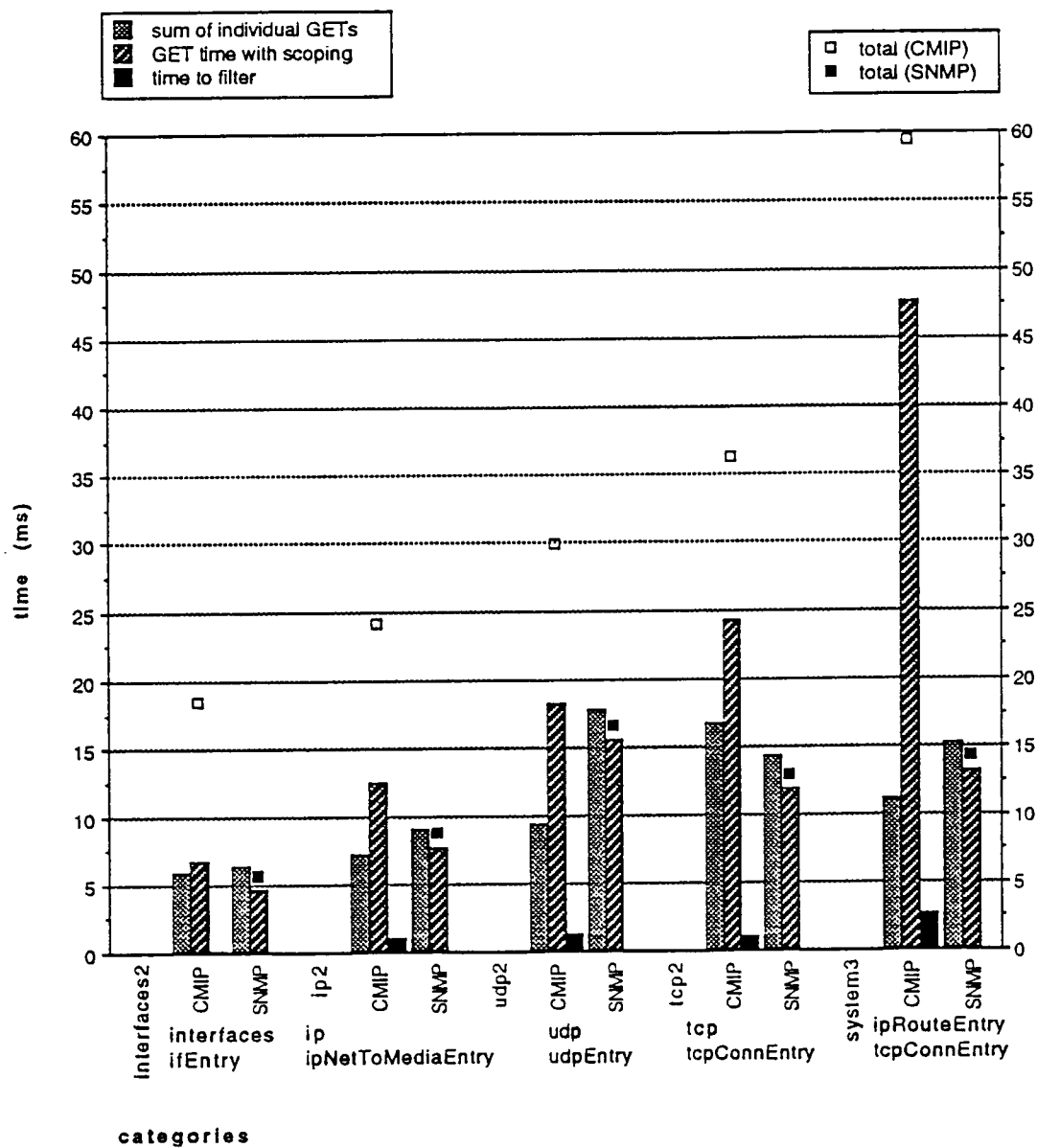


Figure 5.5: CMIP and SNMP data for basic operation 4.

behavior is acceptable or not. Most of the observations made while interpreting the metric 2 data also apply here. The times taken by both protocol engines have decreased from what was observed while evaluating metric 2. For SNMP, this is as not surprising. The reduction in CMIP times is due to the reduction in encoding time as discussed in Section 5.2.3. SNMP no longer displays the large negative scoping overhead for *tcp* since the attribute *tcpCurrEstab* is only retrieved occasionally.

5.2.5 Retrieval Time for the Entire Column of a Tabular Object

Tabular objects are exemplified by the object classes *atEntry*, *ipRouteEntry*, *udpEntry*, etc. Different protocols use different means to traverse a table in order to achieve this function. CMIP uses scoping, whereas SNMP uses its GET-NEXT service in this situation. This metric will clearly outline the efficacy of these varied schemes. Figure 5.6 provides a graphical representation of these metrics for the CMIP and SNMP protocols.

Interpreting the data. SNMP uses the GET-NEXT service repeatedly in order to recover an entire column, whereas CMIP uses its GET service once. With regard to this metric, the following factors determine the performance of a protocol.

1. The decoding time for SNMP is much more as each GET-NEXT request must be decoded. CMIP uses only a single request to get the complete information. Therefore SNMP decoding time is proportional to the number of instances in the target object class.
2. The encoding time for both CMIP and SNMP is proportional to the number of instances being retrieved. However, CMIP uses linked replies. Encoding a linked reply PDU is more expensive than encoding an SNMP reply PDU. This becomes more marked when the number of instances being recovered increases, as is the case with the *udpEntry* (41 instances) object class.
3. The same number of kernel accesses are involved in both protocols. In SNMP however, since the GET-NEXT service is used, the object following the object class being retrieved is also transmitted to the manager at the end. It would therefore be unwise to register two tabular entries consecutively in the MIB. This has not happened in the current SNMP MIB.
4. Since each GET-NEXT request has to be processed individually by the SNMP agent, the time to process the entire column will vary with the number of instances in the table. This is in addition to the decoding time mentioned earlier.

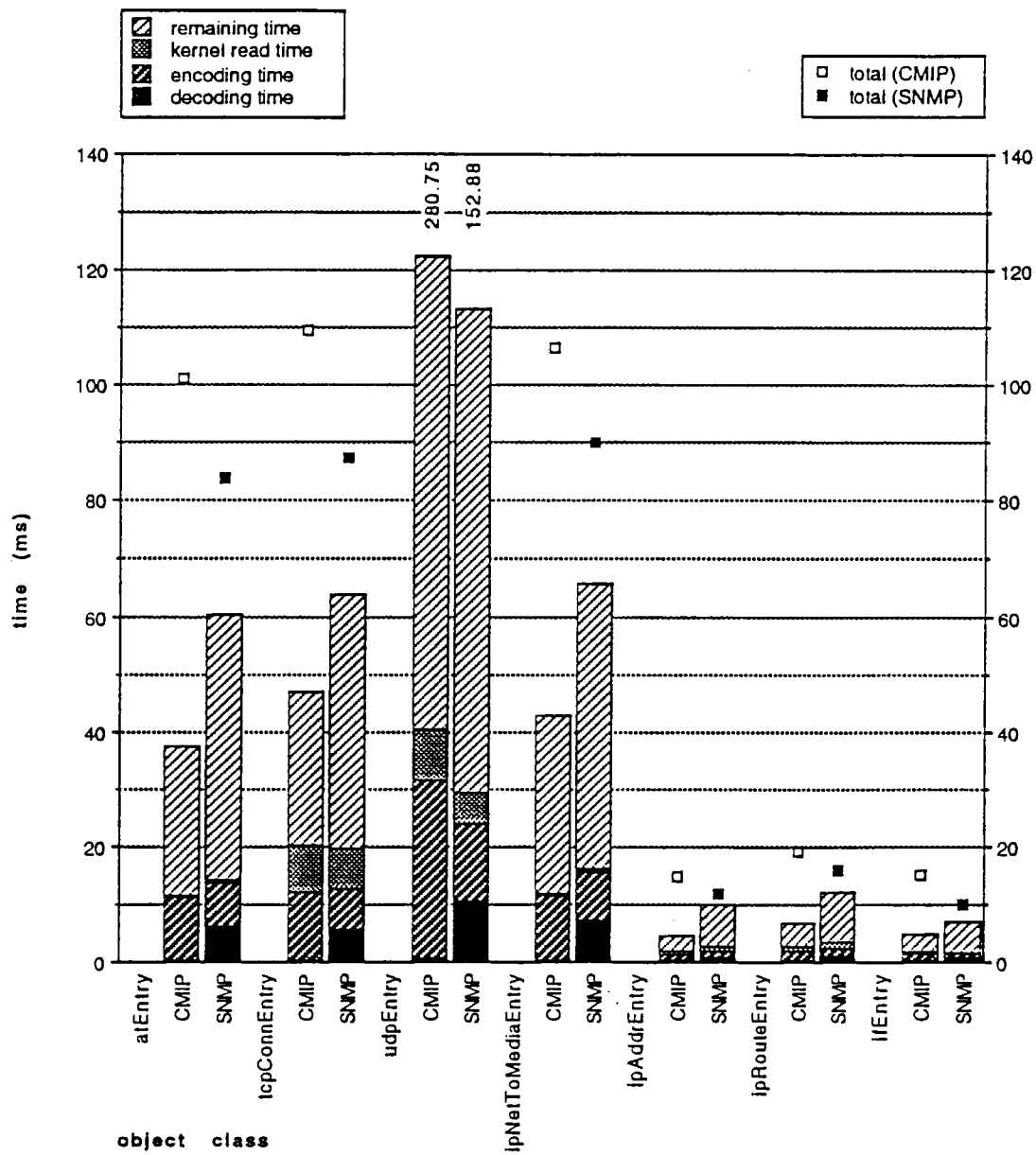


Figure 5.6: CMIP and SNMP data for basic operation 5.

5. The overhead (total time on agent node – time spent in agent alone) increases with the number of instances being recovered as more PDUs have to traverse the underlying stack.

5.2.6 Retrieval Time for Entire Tables

It has often been said that CMIP will do better than SNMP when it comes to the retrieval of complete tables. However, with the kernel data structures currently in use, and the resulting close association of an attribute in a table with the other attributes in its row, extraction of a complete table will be similar to the retrieval times for a single attribute from each row, as reported in the previous metric. On the other hand, this metric would provide information that is quite different from that in metric 5, above, if the individual table rows were to contain information that must be derived from different data structures or different components of the system. Figure 5.7 provides a graphical representation of these metrics for the CMIP and SNMP protocols.

Interpreting the data. The following conclusions may be drawn from the data.

1. The decoding time for SNMP varies with the product of the number of instances being recovered (rows) and the number of attributes (columns) in the object class being recovered. Such dependency is absent in CMIP.
2. The encoding time for both protocols is proportional to the number of attributes present in the object class as well as the number of instances being recovered.
3. SNMP does not perform as well as CMIP since each attribute in the object class has to be dealt with individually in each GET-NEXT operation. This is especially marked for *ifEntry* (22 attributes). The execution time of the SNMP agent is therefore dependent on the number of instances of the object class times the number of attributes in the class. In order to speed up table retrieval with SNMP, pipelined and parallel schemes have been proposed [52]. It has been found that running SNMP over TCP also speeds up the bulk retrieval of entire columns and tables [53].
4. For both protocols, the time spent in the underlying protocol stack is proportional to the number of instances recovered, since that many result PDUs have to traverse the underlying stack.
5. Since all the attributes of a row are kept in the same data structure, the time to recover the entire table is essentially the same as that necessary to recover all

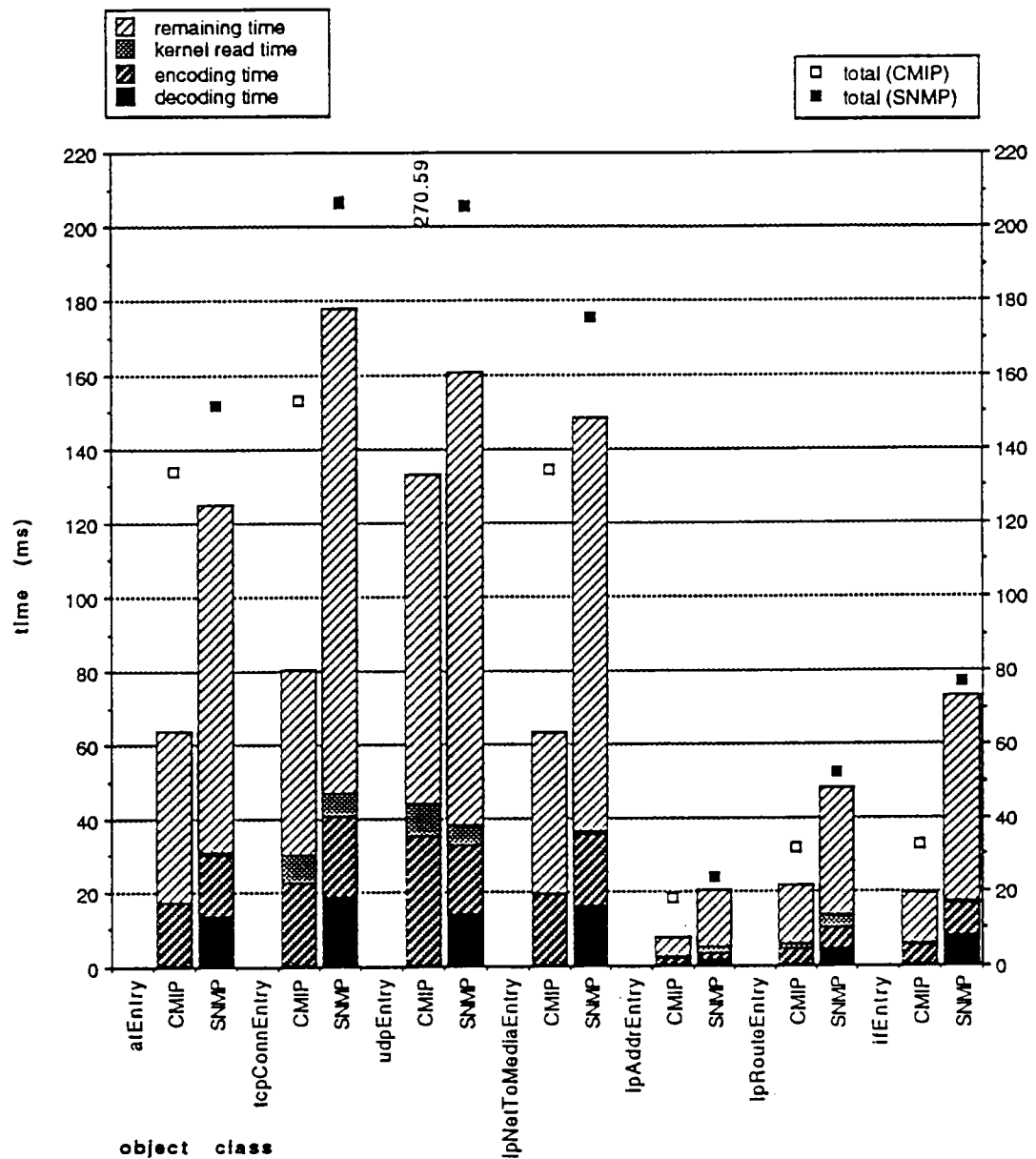


Figure 5.7: CMIP and SNMP data for basic operation 6.

instances of a single attribute, i.e., a full column (metric 5). This is especially true for CMIP. However, the agent times appear greater here because of the time necessary to encode the attribute values into the correct PDU formats which can then be processed by the BER. Additional time is also required to free these PDU structures. This increase is therefore dependent on the product of the number of attributes of an object class and the number of instances of the object class. This increase is evident when the sections marked 'remaining time' in Figure 5.7 are compared.

5.2.7 Time to Set a Single Attribute

Figure 5.8 provides a graphical representation of this metric for the CMIP and SNMP protocols.

Interpreting the data. All the variations seen here are implementation dependent. The system calls required to change *ifAdminStatus* take greater time than what is required to modify *ipForwarding*. Furthermore, there are two instances of *ifAdminStatus* which have to be examined before the modification is made. The differences in time in the CMIP and SNMP environments are also purely due to implementation dependent factors and will not be further elaborated on.

5.2.8 Time to Set a Row Entry

The protocol mechanisms are often somewhat different here. With CMIP, each targeted instance of an object class is specified in the query itself, either explicitly or by the scoping mechanism. All applicable attributes specified in the attribute list are then examined to see if they apply to this instance. In SNMP, the object class is never specified, it has to be deduced from the attributes specified in the query. The other attributes are then checked to see if all the necessary attributes pertaining to this object class have been specified. The operation is completed if that is the case, otherwise an error is returned.

In a comparative evaluation, one must know that some agents may examine the new attribute values provided with the SET directive closely to determine their validity, other agents may trust the manager and perform the operation directly. Though the extensive checking will lower the agent's performance, it is best for the agent to be cautious before carrying out a SET operation. Figure 5.9 illustrates this metric for the CMIP and SNMP protocols.

Interpreting the data. From the data, it cannot be concluded whether one protocol

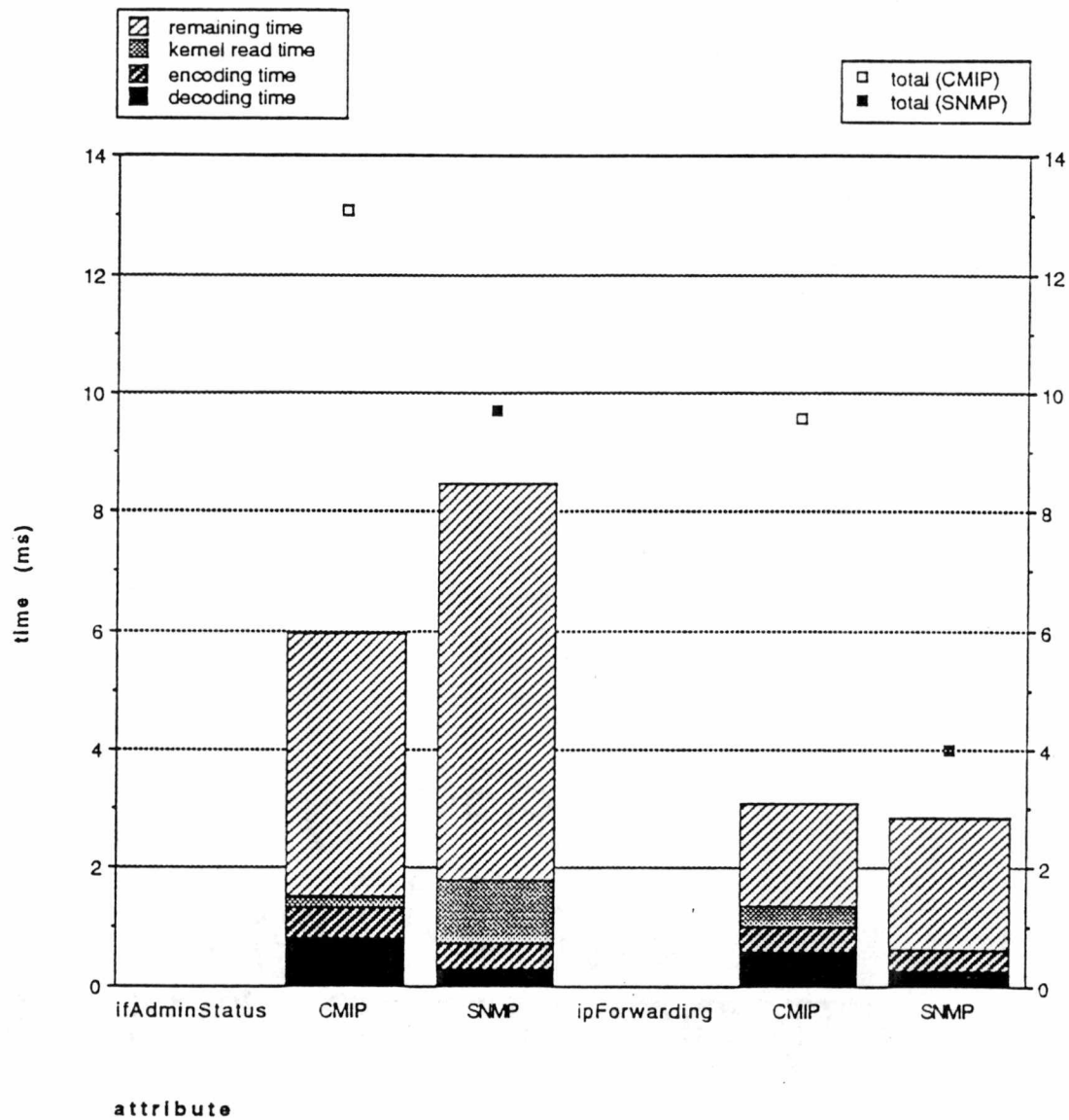


Figure 5.8: CMIP and SNMP data for basic operation 7.

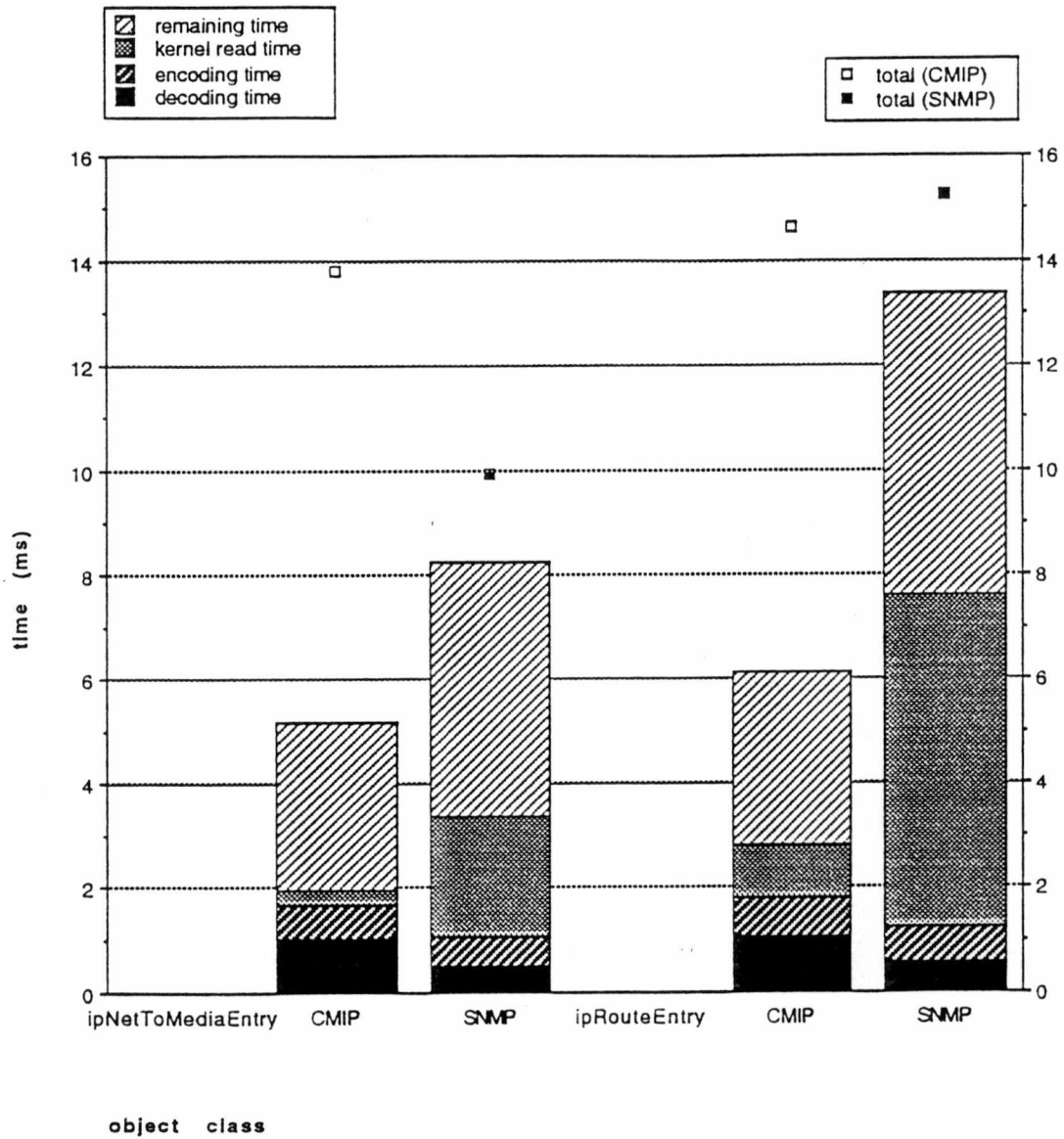


Figure 5.9: CMIP and SNMP data for basic operation 8.

mechanism is quicker than the other, as the extensive checks performed by the SNMP implementation (and its lack in the CMIP implementation) has served to mask that information. In the implementations under test, SNMP utilizes extensive checks to verify the validity of the given data, therefore it takes longer times. Examining `ipRouteEntry` objects is more expensive than examining `ipNetToMediaEntry` objects as more kernel reads are required.

5.2.9 Time to Create an Instance

It is likely that objects will be created or deleted infrequently, hence this metric and the next may not be of major importance to most network managers. They are included here for the sake of completeness. However, the mechanism used to create or delete object instances varies from protocol to protocol and the variation in their performance will be of interest to protocol designers. Figure 5.10 provides a graphical representation of this metric for the CMIP and SNMP protocols.

Interpreting the data. CMIP takes very little time to create an instance, as there is no need to do any kernel fetches. The current CMIP implementation does not perform extensive checks on the given parameters for semantic correctness. SNMP style protocols may take more time to create an instance as the SET command itself is used to create an object. The agent must provide additional processing to support this multiple functionality of the SET operation. All the checks that went into the SET operation are also present here. In fact, SNMP also has to check all existing instances to ascertain whether the SET command requires modification of an existing instance or the creation of a new one. This is expensive due to the associated kernel fetches. Besides, the SNMP agent also has to examine the attribute list in the request and gather all attributes that pertain to the same object class. Recall that SNMP permits attributes to be specified in any order, since there is no notion of object classes. It may be concluded that the protocol mechanism used by SNMP to create a vector object will usually be more time consuming than that used by CMIP — the nature of the object being created will affect this time.

5.2.10 Time to Delete an Instance

The observations made in the previous case apply here. Figure 5.11 provides a graphical representation of this metric for the CMIP and SNMP protocols.

Interpreting the data. Both CMIP and SNMP require to fetch the instance in question before deleting it. This increases the time especially for such objects as

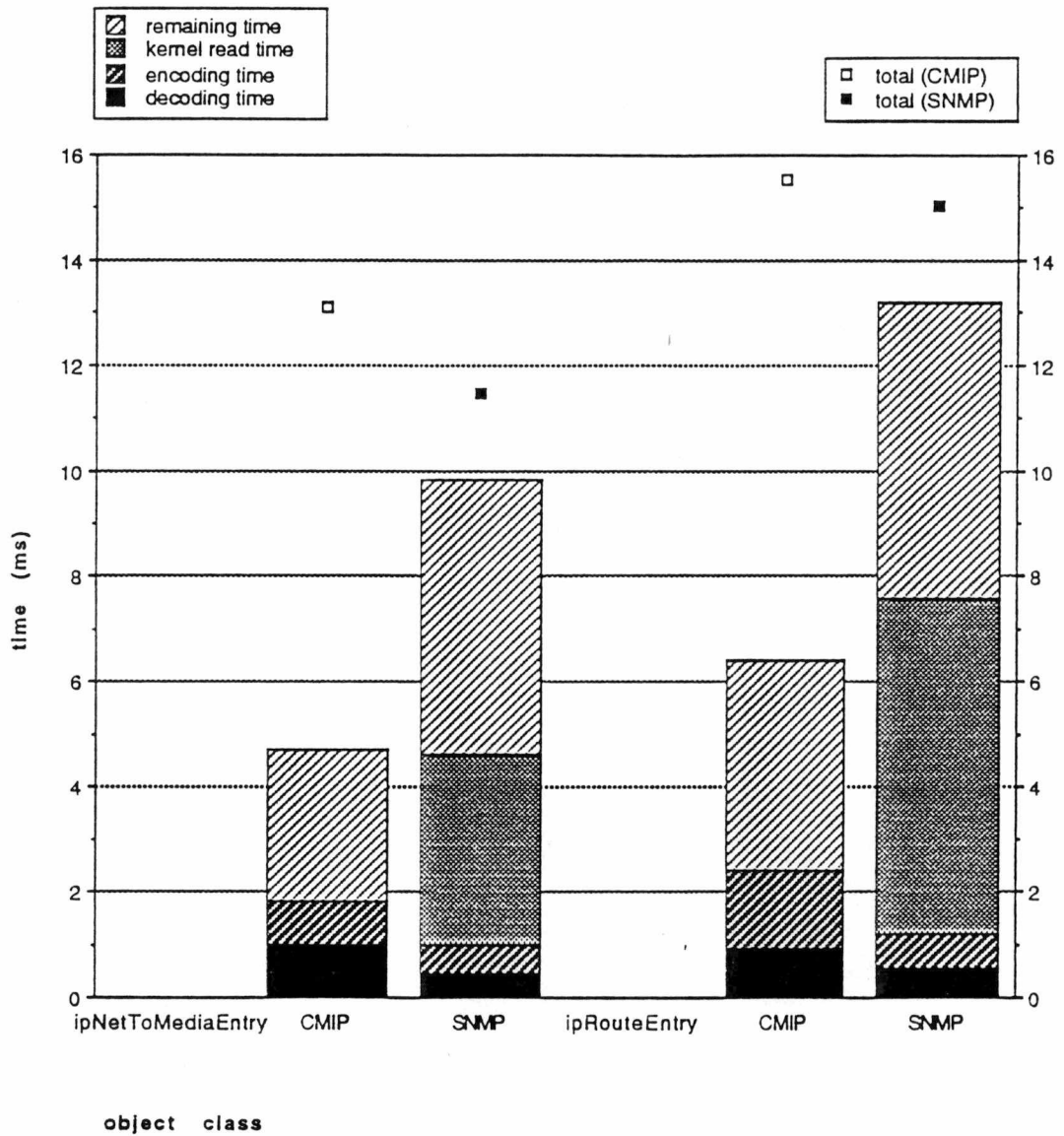


Figure 5.10: CMIP and SNMP data for basic operation 9.

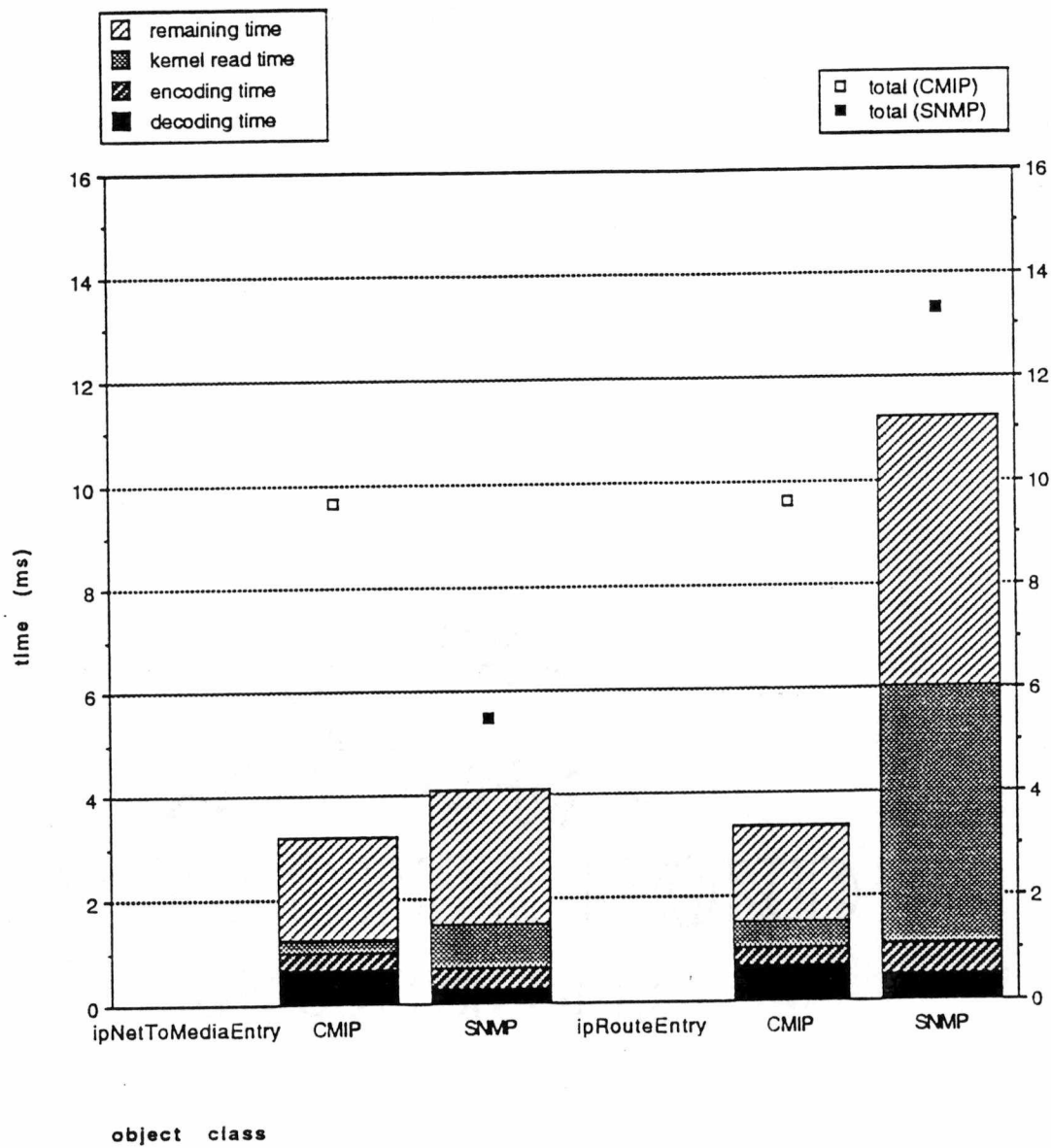


Figure 5.11: CMIP and SNMP data for basic operation 10.

ipRouteEntry. The time to delete an instance of *ipNetToMediaEntry* is relatively low for SNMP, as this is effected just by setting the *ipNetToMediaIfIndex* attribute to 0. When it comes to deleting an *ipRouteEntry* entry, the SNMP implementation not only requires the corresponding *ipRouteType* to be set to *invalid*, but also requires the *ipRouteMetric1* and *ipRouteNextHop* attributes to be specified. Processing the additional attributes takes time as seen in the accompanying figure.

5.2.11 Time to Generate a Trap

Event-reports or traps are central to effective network management. This time will be calculated from the time the trap is detected to the time the agent completes encoding the trap information or PDU. This does not include any processing that may be necessary to detect the trap condition, because in many cases that processing is done periodically, and should be considered part of the protocol overhead. If this processing is done as part of a CREATE, DELETE, or SET operation, it should be included as part of that operation. Figure 5.12 provides a graphical representation of this metric for the CMIP protocol. In this experiment the SNMP implementation generated a trap whenever a bad community name was used. The CMIP implementation generated an event report whenever there was an attempt to set the *sysName* attribute.

Note: The services offered by the trap facilities vary from elaborate (CMIS) to basic (SNMP). Hence, care must be taken while trying to make a relative evaluation. For such purposes, the CMIS event-report service should be configured to functionally match the SNMP trap facilities before a relative evaluation is made.

Interpreting the data. The times for generating the traps are low and very similar for both. This is as expected, as no kernel fetches or other time consuming computation is involved in basic trap generation. The CMIP times will definitely escalate if the full range of CMIS event-report services are used. Only the time spent in the agent itself has been reported in Figure 5.12.

5.3 Research Question 2c: Aspects of CMIP and SNMP Implementations that Lead to Poor Performance

The performance data and the experience gathered during the development of CMIP will be helpful in addressing research question 2c (see Section 1.4). This question deals with determining the aspects of CMIP and SNMP implementations that lead to poor performance.

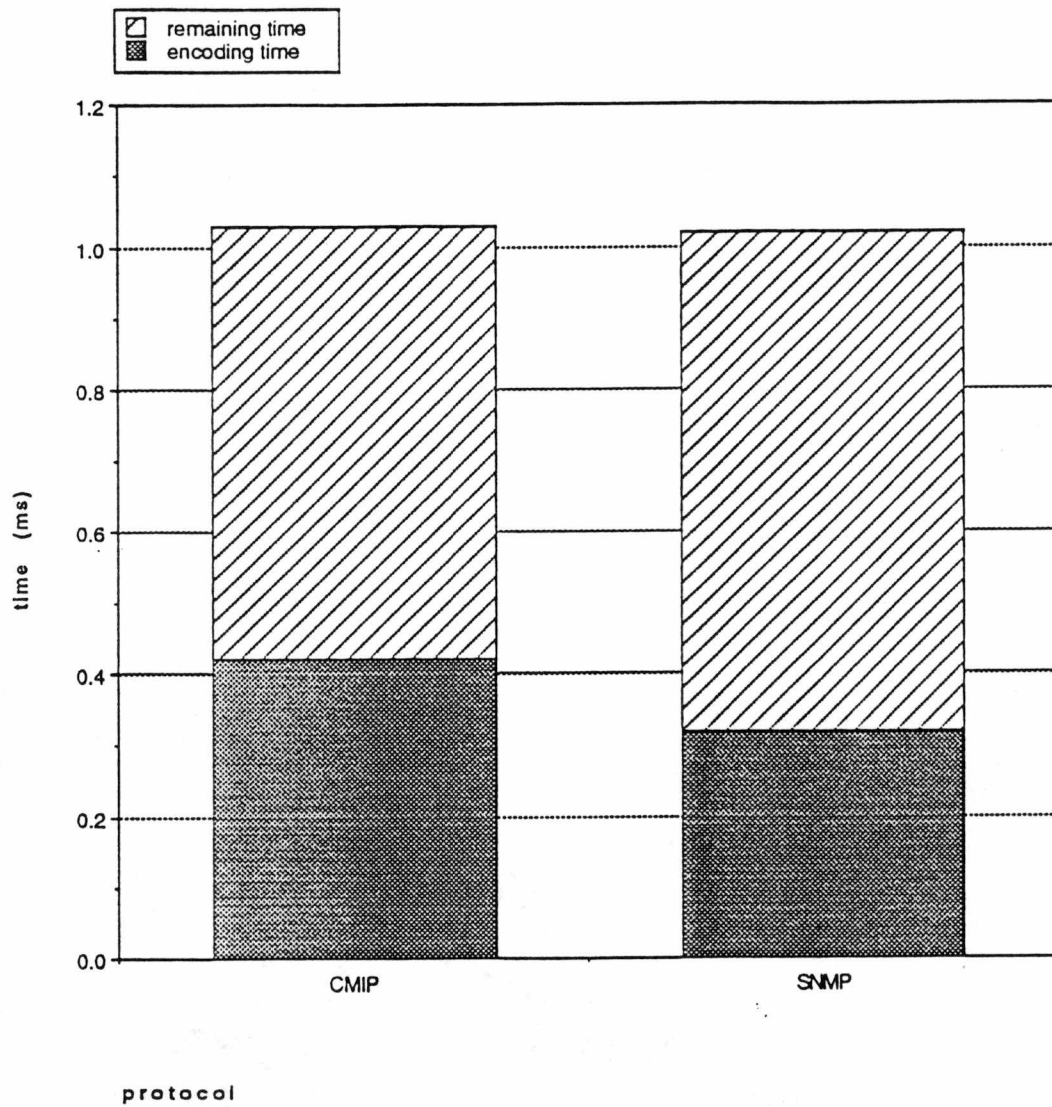


Figure 5.12: CMIP and SNMP data for basic operation 11.

Scoping does not degrade performance when it is used to navigate within a *single object class*. However, as soon as multiple object classes are spanned, performance degradation will occur. This is because every object class (and all its instances) that fall within the scoped subtree have to be examined to see if it matches the accompanying filters. The amount of performance degradation will be proportional to two factors.

1. The number of object instances that fall within the scoped subtree.
2. The amount of processing required to see if the object instance matches the given filter.

Since the first factor is effectively a constant, one can only try to diminish the impact of the second factor. The implementation under test extracts every instance and applies the filter to it. This requires a large number of kernel fetches. The following algorithm could be used to mitigate this situation:

```
Step 1: Extract the attributes present in the filter.  
        Keep them in a list A.
```

```
Step 2: For each object class in the scoped subtree  
        do  
{
```

```
    Step 3: Do any of the attributes in list A  
            belong to this object class ?  
            If NO, go to step 2,  
            else proceed.
```

```
    Step 4: Extract every instance of this object  
            class and apply the filter to it.  
            If the filter matches, perform the  
            requested operation on this instance,  
            else do nothing.
```

```
}
```

Though there is still some processing associated with each object class, this strategy will cut down the number of fruitless kernel reads. This will decrease the scoping ratio as well as the total time spent in filtering (as fewer object instances will be passed through it).

The other option is to limit the scoped subtree to only a few object classes by reformulating the request PDU, so that it would include a list of 'uninteresting' object classes that do not have to be examined. This is not a practical solution at this time as it would mean a change in the CMIP protocol.

Other issues like filtering, encoding and decoding times, etc., also require discussion. Filtering does not take much time. Tests have only included filters of the form:

$$(Attribute1 = Value1) \text{ OR } (Attribute2 = Value2)$$

The diagrams corresponding to metrics 2 and 4 show the times taken by filtering. The total time spent in filtering is proportional to the number of instances being passed through the filter.

It has become clear that for certain object classes performance is closely related to the number of kernel accesses required. Every effort should be made to decrease such access on platforms where management information is kept in kernel memory.

The data also shows that the session, presentation, and application (ACSE and ROSE) layers in ISODE are definitely taking up a lot of time, and may prove to be the greatest bottleneck. Experimenters at University College London (the OSIMIS effort) have also felt this (comment by George Pavlou at the Integrated Network Management II conference, April 1991, Washington D.C.). The penalty is in fact doubled, as delays on both the manager's side and the agent's side add up to determine the duration of the final operation as observed by the manager. This factor is not within the scope of this project and will not be considered further.

The greatest bottleneck in SNMP stems from the fact that there is no way to group related attributes in the object class style when attempting to retrieve or otherwise manipulate them. All attributes have to be specified separately in the query, entailing additional per attribute processing overhead, that is absent in CMIP. The bulk retrieval of tables also constitutes a bottleneck in SNMP, since a separate GET-NEXT query has to be issued for each entry in the table. This requires multiple passes through the lower protocol layers as well as additional encoding and decoding overhead at the manager and agent. As with CMIP, excessive kernel access will be time intensive.

Lexicographic ordering, which requires the sorting of tabular objects being retrieved, has not been observed to be unduly time consuming.

5.4 Conclusions

A network management protocol is a provider of different services rather than a single monolithic entity. The services exhibit different performance characteristics when applied to different basic operations. The basic operations are therefore the invariants in a network management scenario. The performance metrics are therefore the times required to perform predefined basic operations on a cross section of managed objects.

The underlying protocol layers of the OSI protocol stack will greatly degrade the performance of any management protocol running over the OSI application layer. CMIP is also prone to a drop in performance when a scoping operation is performed over multiple object classes. The simple filters examined in the current implementation consume negligible time.

The use of lexicographic ordering in SNMP means that entire tables often must be recovered and sorted before a response is generated to a GET or GET-NEXT request. This is time consuming if the table resides in kernel memory.

The performance data shows that object oriented MIB design may increase or decrease the encoding and decoding times on both the manager and the agent depending on the query being used. This happens because the PDU size is affected by the object oriented approach as well as the type of the query. The small size of the current MIB does not allow evaluation of the capabilities of the ISO SMI when applied to a larger, more complex MIB.

Figure 5.13 shows that moving filtering to the manager will lead to increased load on the manager, agent, and the network. The encoding and decoding to be done by the manager and the agent will increase. The performance data shows that it takes longer to encode and decode information than it does to filter. Therefore, moving a filter function from the agent to the manager will be detrimental to system performance, unless this function is extremely resource intensive.

A problem with evaluating protocol performance is that many of the impediments to execution speed have to do with implementation, and are not directly related to the protocol specification itself [69, 72]. This is particularly true when the service aspect of a protocol dominates its message exchange aspect, as is the case with network management protocols. Analytical performance models that can sometimes be derived from the protocol specification itself are of limited value in such a situation, and greater importance has to be placed on experimental observations such as those presented here.

Finally, no single performance metric will suffice to evaluate a network management

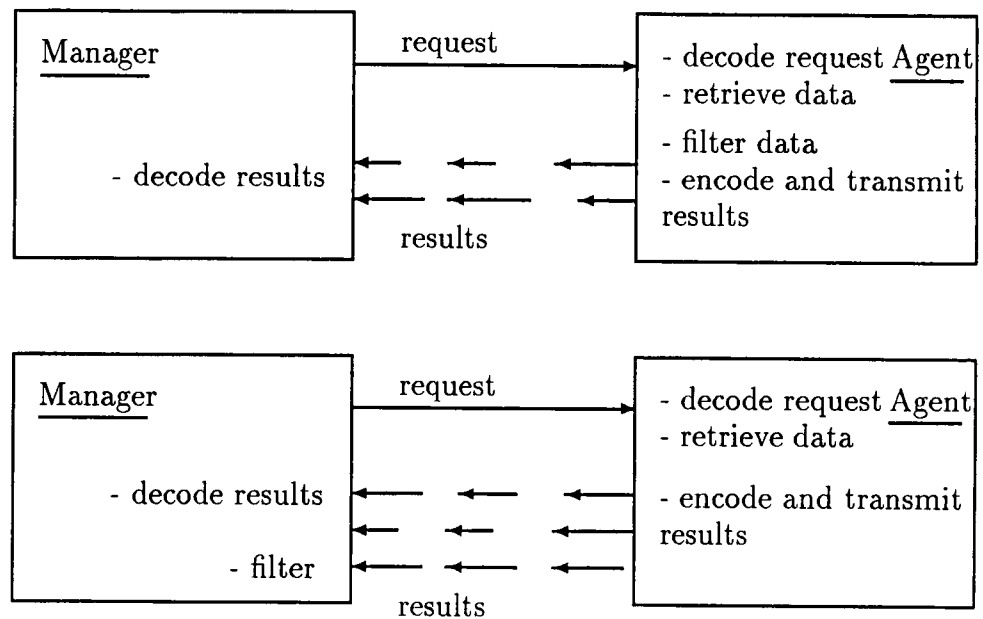


Figure 5.13: Filtering done on the agent (top) and on the manager (bottom).

protocol. They should be considered together. If an overall verdict on the performance of any network management protocol is required, it will depend on the metrics on which the evaluator places importance, and on what the manager expects from the management facility.

Though beyond the scope of this project, the tests that have been formulated here can also be used to evaluate different implementations of the same protocol. This will be a simpler task, since there will be no need to give special attention to implementation dependent effects in the data before conclusions can be drawn.

Chapter 6

Evaluation of the Agent's Memory Requirement

6.1 Objectives

The object is the evaluation of the agent's memory requirement. It has been observed in Section 3.2 that the memory requirements of the agent can be best analyzed if it is subdivided into its constituent text, heap, data, and stack segments.

The objectives of this chapter are:

1. measurement and analysis the agent's text segment,
2. measurement and analysis the agent's heap segment,
3. measurement and analysis the agent's data segment,
4. measurement and analysis the agent's stack segment, and
5. to address research questions 2b and 2d.

6.2 Controlling the Experimental Environment

The following factors should be controlled during every experiment.

1. System Load

The existence of other processes on the agent node will not directly affect the memory required by the agent. However, the number of entries in the tabular objects may influence the amount of memory used, because some protocols may

require the existence of the entire table in its process space in order to perform some protocol specific functions on them. For instance, an SNMP agent will find it convenient and time saving to fetch an entire table before sorting the constituent entries in lexicographic order. This lexicographic ordering is required for SNMP's GET-NEXT service. The point is that the *udpEntry*, *tcpConnEntry*, and *atEntry* tables may change in length if other processes run on the agent machine, thus affecting the memory requirement of the agent. Hence it is recommended that other processes not be run on the agent node during the experiments. Table 6.1 shows the number of instances of the tabular managed objects that were present during the performance measurements.

Table 6.1: The number of instances for the tabular objects.

Object Class	Number of Instances	
	CMIP	SNMP
<i>ifEntry</i>	2	2
<i>ipAddrEntry</i>	2	2
<i>ipRouteEntry</i>	6	6
<i>atEntry</i>	12	12
<i>ipNetToMediaEntry</i>	12	12
<i>tcpConnEntry</i>	26	26
<i>udpEntry</i>	40	41

2. The amount of logging done by the agent

Logging activities should be avoided since the standard I/O library requires heap space for its operation.

3. The computing environment

All experiments should be conducted in the same computing environment in order to avoid possible variations in the agent's memory requirements.

6.3 The Text Segment

6.3.1 Measurement Strategy

The text and data size of a process remain essentially constant during its running time. Therefore, they can be obtained by examining the name list (sorted by address)

provided by the `nm -n` shell command. The `size` command is not suitable for this purpose because the size reported by it also includes material that is introduced by the compiler and the loader, material which does not belong to the actual program being measured.

Analysis of the text segment involves the determination of how much text is required for particular sections of a protocol. This will permit one to pinpoint those aspects of a protocol that require large amounts of code and accompanying storage.

For ease of analysis, an agent's code can be subdivided into the groups outlined below. This division assumes that the ISODE stack is linked statically to the agent and will therefore contribute to the agent's size. The classification has been influenced by the CMIP and SNMP agents that are providing an experimental basis to the current study. It is expected that other protocols or implementations can be subdivided in a similar manner. All the sections shown here may not apply to every protocol or implementation.

The Components of a Network Management Agent

Experience with current network management protocols indicate that a management agent can be approximately subdivided into the following sections.

1. Association control.
2. Remote operations.
3. Initial checks to ensure the validity of the received association establishment PDU and starting up the agent daemon.
4. Encoding and decoding the PDUs:

Applications that use any form of abstract syntax for transferring data from one node to another, may do it in two phases [3, p421]. Initially, the native data structures (those that can be used readily by the application) are converted to an intermediate syntax. In the second phase, these intermediate structures are converted to the octet strings that are to be transmitted. This terminates the encoding process. The process is reversed during decoding. The following grouping is therefore necessary:

- (a) Encoding a native structure to the intermediate structure.
- (b) Decoding the intermediate structure to a native form.

- (c) Encoding the intermediate structure to an octet string suitable for transmission.
 - (d) Decoding an octet string into the intermediate structure.
- 5. Functions to free the intermediate data structures mentioned above.
- 6. Functions to display the intermediate data structures mentioned above.
- 7. Functions to free native data structures.
- 8. Functions to display native data structures.
- 9. Support for scoping.
- 10. Support for filtering.
- 11. Support for the following management operations:
 - (a) GET
 - (b) GET-NEXT
 - (c) SET
 - (d) CREATE
 - (e) DELETE
 - (f) EVENTS/TRAPS
 - (g) Miscellaneous support functions for these operations.
- 12. Data conversion routines. These include functions that are used extensively to convert data from one form to another during encoding and decoding.
- 13. Support for communities, MIB views, and proxy functions. Note that proxy functions are not part of a basic management agent. It is an additional service that may be provided.
- 14. MIB initialization and other MIB related actions, i.e., searching for a particular node in the MIB tree.
- 15. MIB traversal. Since the MIB is often organized as a tree, protocols would require to search this tree in some fashion, especially for scoping or a GET-NEXT type of operation.
- 16. Initialization of the different agent groups.

17. Presentation layer services.
18. Session layer services.
19. Transport layer services.
20. Functions that bridge the gap between the ISODE transport layer functions and the underlying socket interface of BSD Unix.
21. Other functions.

6.3.2 OSI Remote Operations

Readers outside the OSI community may not be familiar with the Remote Operations Service (ROS) that is available to applications using the OSI stack. Since ROS related operations will take up a significant amount of CMIP agent code, this section will provide a brief outline of ROS. The Remote Operations Service Element (ROSE) is essentially a facility for supporting remote procedure calls. Of special interest here is the RO-notation which is used to write a formal specification of the remote operations that are allowed by any particular application. The ASN.1 specification of the arguments and returned values or error conditions that correspond to the remote operations are also part of the RO-notation module of an application. Since the description is in ASN.1 it is possible to automatically generate functions that encode and decode these data types from an intermediate high level form to a series of octets that may be transmitted and vice versa. The intermediate structures can be converted by local non-standard means to native data structures that are usable by the application. Routines can also be generated automatically to free or display these intermediate structures. ISODE provides a tool called Pepy, which is an acronym for Presentation Element Parser (Yacc based), that automatically generates routines that convert data between the BER encoded form and an intermediate form. Another ISODE tool called Posy generates the intermediate data structures from their ASN.1 specifications. The preceding ideas are necessary to interpret the data on the CMIP agent's text size since the CMIP agent under test makes extensive use of the Pepy generated functions. Although ROSE does offer some other services like the generation of stub routines for invoking remote operations, they will not be elaborated on here. An overview of ROSE may be found in [3].

6.3.3 Data and Analysis

The Text Segment of the CMIP Agent

Table 6.2 provides data on the text section of the CMIP agent. Total figures are rounded off to the nearest Kbyte since the text segment will usually cover an integral number of pages. Percentages will also be rounded off since the trend is important, not extreme accuracy. It is interesting to note that a large section of the text is taken up by the upper layers of the OSI protocol stack, i.e., the application, remote operations, presentation, and session layers (items 1, 2, 17, and 18 in Table 6.2). They make up 29% of the code. This may not be a problem on a powerful host running a number of OSI applications, for then this code could be provided as a shared library or placed in the kernel of the operating system. However it may be a high overhead when one tries to introduce CMIP to intermediate systems which would ordinarily only need to support the three lower layers of the protocol stack.

The code necessary to encode, decode, free, and display the structures generated automatically from the RO-notation specification of CMIP take up another 22% of the total code (items 4c, 4d, 5, and 6; fourth column). The routines for displaying the structures may be dropped from the agent if necessary, since they are not critical to the agent's operation and serve chiefly as debugging tools. In sharp contrast, similar routines for the SMI and the MIB together take up only 12% of the code (items 4c, 4d, 5, and 6; third column). This is because the SMI and MIB require fewer and simpler ASN.1 structures than the RO module.

Proceeding to the actual management operations, it is seen that the GET operations require more code than the other operations. A greater number of object classes support the GET operation, hence this result. Strangely enough, though there have been some misgivings about the complexity that scoping adds to CMIP, no explicit code is necessary to support it. Scoping is woven into the recursive MIB tree traversal routines.

A drawback of current coding techniques is the amount of conversion necessary between native data representations and those necessary to translate the data into transmission ready octet strings. Item 12 in Table 6.2 clearly points this out. This is the penalty one must pay to bridge the **semantic gap** between standardized physical byte streams that are transmitted on the wire and the data that can be used conveniently by an application. Closing this gap would be a worthwhile endeavor in itself.

The OSI transport layer, together with the routines that map the OSI transport services to the underlying TCP interface takes up a sizable part of the code. However in a pure OSI stack, the mapping functions will not be necessary, because the OSI

Table 6.2: CMIP agent text size.

no.	agent component	size in Kbytes		
1	association control	53.07		
2	remote operations	37.32		
3	agent start up	3.34		
4a	encoding: native → intermediate	5.10		
4b	decoding: intermediate → native	9.71		
		SMI & MIB	RO module	common
4c	encoding: intermediate → octets	16.04	35.34	2.21
4d	decoding: octets → intermediate	27.29	47.77	3.02
5	free intermediate structures	5.75	12.07	0.42
6	display intermediate structures	24.84	44.30	2.96
7	free native structures	none		
8	display native structures	25.80		
9	scoping	no explicit routines		
10	filtering (limited complexity)	1.80		
11	the management operations			
a	GET	19.17		
b	GET-NEXT	not applicable		
c	SET	8.17		
d	CREATE	8.51		
e	DELETE	2.89		
f	EVENTS (limited complexity)	0.88		
g	supporting functions	1.94		
12	data conversion routines	27.88		
13	communities, MIB views, proxy	not applicable		
14	MIB initialization	12.87		
15	MIB traversal	1.78		
16	agent group initialization	2.93		
17	presentation layer	61.69		
18	session layer	31.48		
19	transport layer	56.70		
20	TSAP ↔ TCP/IP	35.39		
21	others	4.29		
TOTAL		635		

transport layer would function directly over the OSI Network Service Access Point (NSAP).

Coming back to the encoding and decoding required by the agent — the fact that the CMIP SMI does not limit the ASN.1 syntax that may be used to specify the MIB has been a source of some concern, since more code would be necessary to parse all the possible ASN.1 structures. It has been observed that these fears are unfounded, since the CMIP MIB employed in the current study is essentially identical to the Internet MIB and therefore uses a limited number of ASN.1 structures. As noted earlier, the code necessary to process them is quite modest. This will change as the complexity of the MIB increases. However, the variety of ASN.1 structures defined in the ROS specification exceeds the number of different ASN.1 structures defined in the SMI. Hence more code is required to process the ROS specification of CMIP.

The considerable size of the CMIP RO specification related components (22% of the code) leads one to two viable conclusions:

1. The CMIP RO specification uses too many different ASN.1 structures that require encoding, decoding, and other processing.
2. The automatic code generator (Pepy) produces bulky code.

Changing the RO specification would be a drastic step, because it entails a change in the protocol itself. Such a step can only be considered after more experience with CMIP is gathered, especially after its deployment on low memory network components. It is well known that Pepy generates voluminous code. However, the volume of generated code has decreased with the introduction of newer code generation tools like Pepsy [56–57]. Whether it will decrease sufficiently to warrant the current complexity of the CMIP RO module remains to be seen. It should be noted that this decrease in generated code size may often be at the cost of elevated running times.

The Text Segment of the SNMP Agent

Table 6.3 presents the SNMP text size data. The SNMP text size is remarkable because of its smallness in relation to the CMIP agent's size. This is largely because SNMP does not use the upper layers of the ISO stack or the RO notation. The other principal point of interest is the small amount of code necessary to decode and encode the SNMP PDUs. This is a direct consequence of the fact that SNMP has only two separate PDU structures, one for retrieving, setting, and responding, the other for reporting traps. *This elegant use of the same PDU structure for multiple operations*

Table 6.3: SNMP agent text size.

no.	agent component	size in Kbytes
1	association control	not applicable
2	remote operations	not applicable
3	agent start up	9.20

4a	encoding: native $\rightarrow$ intermediate	2.29
4b	decoding: intermediate $\rightarrow$ native	2.46
4c	encoding: intermediate $\rightarrow$ octets	5.70
4d	decoding: octets $\rightarrow$ intermediate	9.20

5	free intermediate structures	2.77
6	display intermediate structures	2.38
7	free native structures	1.06
8	display native structures	1.29
9	scoping	not applicable
10	filtering	not applicable

11	the management operations	
a,b	GET/GET-NEXT	27.17
c	SET	20.00
d	CREATE	not applicable
e	DELETE	not applicable
f	TRAPS	1.63
g	supporting functions	nothing explicit

12	data conversion routines	18.28
13	communities, MIB views, proxy	15.23
14	MIB initialization	10.69
15	MIB traversal	nothing specific
16	agent group initialization	10.00
17	presentation layer	not applicable
18	session layer	not applicable
19	transport layer	38.35
20	TSAP $\leftrightarrow$ TCP/IP	36.03
21	others	18.00
TOTAL		232

has greatly decreased the amount of code necessary to process PDUs. It is now felt that SNMP could be designed with just one PDU type rather than two. A separate Trap-PDU would then be unnecessary.

The SET service appears expensive because it performs extensive checks before proceeding with the requested operation. In addition, it also incorporates the CREATE and DELETE services in the same code. The latter services are handled separately in CMIP.

In the SNMP implementation under study, 32% of the text is made up of the OSI transport layer and the accompanying routines that will map the OSI transport service directives to corresponding UDP operations. The use of the ISODE for this implementation has introduced this overhead.

6.3.4 Closing Remarks

The text sizes of a CMIP and an SNMP agent have been presented here, along with the conditions under which these measurements were made. These results show the agent sizes that can be expected for these protocols. The coding tools and the implementation strategy will play an important role in determining the absolute size of the agent in any given situation. The lower layers of the ISO protocol stack contribute heavily to the CMIP agent size. The large number of ASN.1 structures introduced by the RO specification of CMIP also requires a significant amount of code to process. An important finding is that wide semantic gap exists between the BER encoded physical data representation and the data format that can be conveniently used by the agent at the application layer. Item 12 in Tables 6.2 and 6.3 clearly indicates that closing this gap requires a large number of data conversion routines. Bridging this gap is an important problem that is yet to be solved satisfactorily. Surprisingly enough, if the overheads imposed by the underlying protocol stack, the ROS, and the differences in the amount of coding and decoding necessary are ignored, the two agents will not differ vastly in their size.

6.4 The Heap Segment

6.4.1 The Workload

The basic management operations used in the performance evaluation experiments serve as a set of typical, elementary operations, a combination of which can be used to carry out any management task. The same operations will therefore be used here

to evaluate the heap and stack requirements of the agent modules. The techniques to be used for these measurements make the use of multiple repetitions of an operation unnecessary (as opposed to the 2000 repetitions used in the performance evaluation). However, in each category, a mix of different instances of the targeted attribute or object class is used in each measurement session. Though this is not expected to affect the memory usage data greatly, it is nevertheless sound practise to do so. It may even prove to be crucial in some situations.

6.4.2 Measurement Strategy

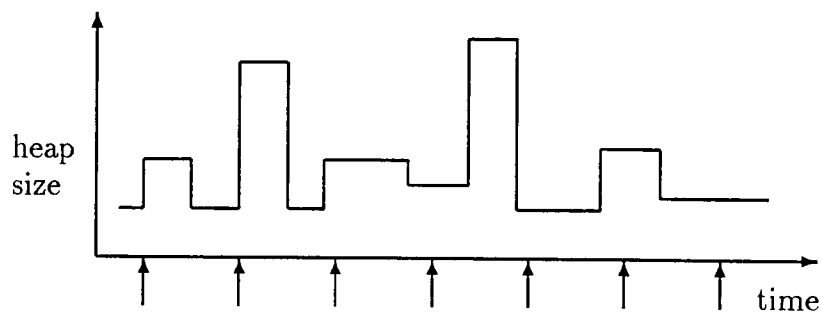


Figure 6.1: Sampling occurring at times when the heap size is sub-maximum.

The heap size changes dynamically. It must be monitored continuously in order to determine the maximum heap size. The lowest sampling period on the Sparcstation 2 is 10 ms. Since this is a long period compared to the running times of many of the functions used by a typical management agent, the interrupts for sampling may occur just before or after memory has been allocated, utilized, and freed. This will cause one to miss the peaks of the heap size profile. Figure 6.1 illustrates this hypothetical situation. In the figure, the vertical arrows mark the sampling intervals.

The following technique has been adopted to solve this problem. Since the size of the heap is controlled by the `malloc()`, `calloc()`, and `free()` functions, `malloc()` and `free()` were instrumented to keep a running count of the heap size as each of these functions were called. `Calloc` simply places a call to `malloc()` and therefore requires no additional instrumentation. The agent was linked to these modified functions instead of to the standard memory allocation functions provided by the C library. This ensured that the other routines contained in the kernel and the C library would also utilize these modified functions.

An operation's heap requirement may now be measured by executing the operation

once only. There is no need for multiple repetitions. This is a tremendous advantage in terms of reducing the run time of the experiments.

The maximum heap size (MHS) can be calculated by instrumenting the `malloc()` and `free()` functions. Though the MHS is necessary to determine the total amount of heap space required by a particular implementation of an agent, it will not permit a complete analysis of the agents memory (heap) requirements due to the following limitations.

1. The MHS can be, and is, influenced by factors that are external to the protocol itself. For instance, the C library function `fgets()` is used widely by implementations to read agent configuration files. On a Sparcstation 2, running SunOS 4.1.1, this function may require a cache of 8 KBytes, since that is the recommended block size for file system I/O on the system. Such an incidental allocation can raise the MHS to a level above that which is required by the protocol mechanism alone. Once the MHS becomes latched to a high value, any further allocation/deallocation activity that occurs below this level will not affect the MHS and would therefore go unnoticed. On the other hand, it is possible for protocol related allocations to outstrip the requirements imposed by the incidental use of functions such as `fgets()`. Figure 6.2 illustrates these situations. Here the allocation between t_1 and t_2 is an incidental allocation. In the top figure, the incidental increase in heap overshadows protocol related allocations. In the lower figure, the protocol related allocations nullify the effect of an incidental allocation. The point is that the MHS alone is a poor indication of a protocol's actual heap requirement.
2. Since the aim is to evaluate a protocol, it is desirable to know how much heap space is required for different aspects of a protocol, like building an in-core MIB tree from the standard ASN.1 MIB specifications, or encoding the reply to a management query. The MHS alone will not provide this information.

Both difficulties can be removed by a detailed look at the agent's memory allocation/deallocation pattern. The method proposed here is simple, albeit tedious.

Obtaining the agent's memory allocation/deallocation pattern and correlating it to particular sections of the agent's code.

Step1:

The `malloc()` function is instrumented with a statement that will output the amount of memory requested, the amount of heap space the process currently holds, and the

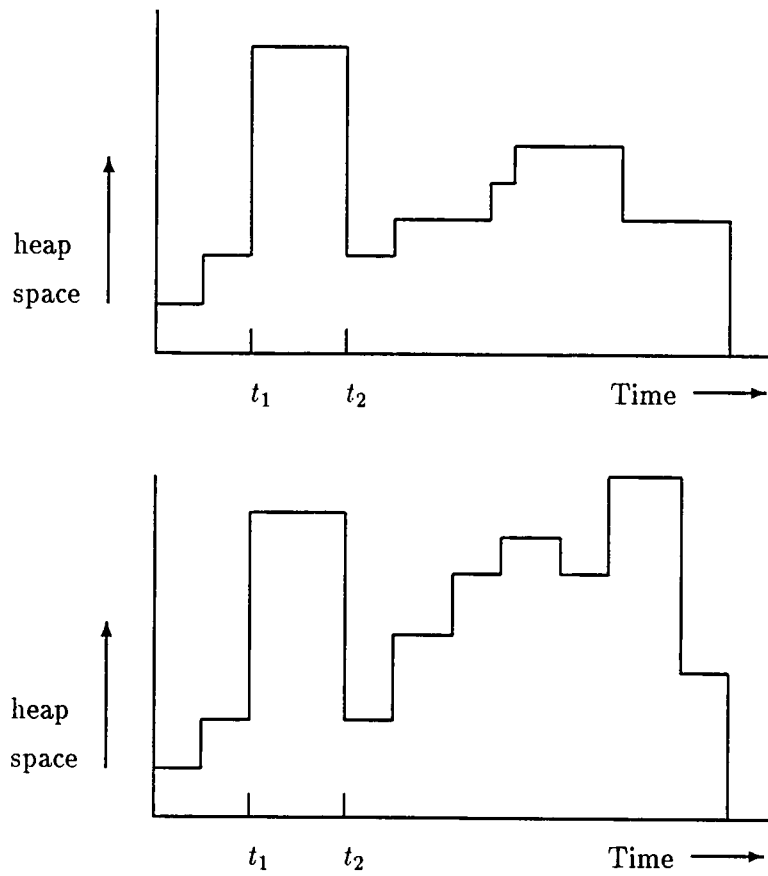


Figure 6.2: The effect of incidental allocations on the maximum heap size.

maximum heap size achieved by the process so far. The catch is that functions like `printf()`, etc., that are part of the `stdio` library should not be used, because they themselves include calls to the allocation functions, and this would therefore lead to infinite recursion. The `malloc()` function in the C library is accompanied by a special `sprintf()` function which avoids the `stdio` library. This has been used to output the necessary information.

Step2:

Checkpoints are placed at strategic points within the agent code. This is simply an output statement which will output to the same device as the instrumented `malloc()` function. Once more, functions from the `stdio` library should be avoided while implementing the checkpoints. Since they use the memory allocation functions, their use may change the MHS readings, though it would be by a negligible amount. What is of greater interest is the fact that the instrumented `malloc()` function directly uses the `write()` system call to achieve `stdio` independency. The agent should also use this function for the checkpointing in order to ensure that the statements that mark the checkpoints and the output statements in `malloc()` are printed out in the proper order. The use of `printf()` may destroy this order because `printf()` and `write()` use different output mechanisms, hence intermixing the two would may produce results not in keeping with the relative order of those statements in the program flow. Consider the following section of code:

```
...
printf("Checkpoint 1\n");
...
p = malloc(20);
q = malloc(50);
...
free(q);
printf("Checkpoint 2\n");
...
r = malloc(40);
printf("Checkpoint 3\n");
...
```

Assuming that the `malloc()` function is instrumented with the `write()` function, the result of running this code might be:

```

...    ...
request = 20, current heap size = 20, MHS = 20
request = 50, current heap size = 70, MHS = 70
request = 40, current heap size = 60, MHS = 70
Checkpoint 1
Checkpoint 2
Checkpoint 3
...    ...

```

This would give the observer the mistaken notion that all the allocation occurred before checkpoint 1. This situation may be remedied by using `write()` statements in the checkpoints rather than a `printf()`. This would guarantee output of the form:

```

...    ...
Checkpoint 1
request = 20, current heap size = 20, MHS = 20
request = 50, current heap size = 70, MHS = 70
Checkpoint 2
request = 40, current heap size = 60, MHS = 70
Checkpoint 3
...    ...

```

The correct trace of the memory allocation pattern will tell the observer which sections of code are invoking the memory allocation functions. Finer granularity can be obtained by using one of many existing tools that allow one to step through every line of the code (*dbx* is one such tool). If the area between checkpoints 1 and 2 is of interest, then one can step through the code included therein and find out exactly where and why an allocation takes place. Checkpointing, together with the use of an available tool like *dbx* solves the second problem mentioned previously.

However, debugging tools like *dbx* cannot be used to step through code that is only available in object form or as a library. Such code essentially remains a black box unless one goes through the corresponding source files manually in an effort to find where dynamic memory allocation takes place. The same applies to agents that detach from a parent as a child process. Such child processes cannot be stepped through, but they can be checkpointed.

The question of determining which allocations are purely a quirk of the implementation (and perhaps occurring without calculation) is a subjective one. A heuristic approach has been taken here. It has been observed that most allocations ask for

relatively few bytes, typically less than 100 bytes. As a result requests for relatively large amounts of memory (exceeding 500 bytes for instance) will easily stand out in the memory allocation trace. These large requests are then to be investigated. Once their cause is known, the evaluator can make a decision on whether that allocation is incidental or not. Excluding the smaller allocation requests from this heuristic screening is justifiable because either way, a few small incidental allocations will have little or no affect on the MHS or on any analysis of the agent's memory requirements.

These methods have been applied to locally available implementations of the CMIP and SNMP protocols. For each protocol, the results have been partitioned into the following categories:

1. **The maximum heap size for individual queries.** This is necessary to determine how much memory is required to run the agent, and it is especially important for diskless platforms that cannot take recourse to virtual memory methods. The effect of individual queries on the heap requirements is shown.
2. **The heap size at 'idle state'.** This refers to the size of the heap when the agent is listening for queries. Agents will spend most of the time in such an idle state. In platforms equipped with virtual memory, the entire agent process can be swapped out during these periods. However, on diskless platforms the agent is resident in main memory at all times. In these situations it is important to know how much space (in this case heap space) is taken up by an idling agent. This space is pure overhead because the other processes on the system will not be able to access that space.
3. **Analysis of requests for heap space.** This is an examination of the agent's heap request trace and will examine the heap requirements of separate aspects of the protocol, e.g., agent start up, MIB tree creation, etc.

6.4.3 Data and Analysis

The Heap Segment of the CMIP Agent

All heap measurements are be rounded off to the nearest Kbyte because heap space is allocated in blocks that are usually multiples of a Kbyte.

The Maximum Heap Size for Individual Queries

Table 6.4 lists the maximum heap sizes for individual queries. The MHS_{full} column

Table 6.4: CMIP agent heap sizes.

OBJECT CLASS OR CATEGORY		MHS_{full} (Kbytes)	MHS_{mod} (Kbytes)
1.	Retrieving a complete object class		
i.	atEntry	36	24
ii.	icmp	38	30
iii.	ipAddrEntry	36	24
iv.	interfaces	36	23
v.	ipRouteEntry	36	26
vi.	ip	36	28
vii.	udp	36	24
viii.	tcp	36	26
ix.	tcpConnEntry	36	25
x.	ifEntry	38	30
xi.	ipNetToMediaEntry	36	24
xii.	udpEntry	36	24
2.	Retrieving an instance each from different object classes		
i.	interfaces2	38	30
ii.	ip2	38	30
iii.	udp2	36	24
iv.	tcp2	36	28
v.	system3	47	39
3.	Retrieving a single attribute from an object class		
i.	atEntry	36	23
ii.	icmp	36	24
iii.	ipAddrEntry	36	23
iv.	interfaces	36	23
v.	ipRouteEntry	36	24
vi.	ip	36	24
vii.	udp	36	23
viii.	tcp	36	23
ix.	tcpConnEntry	36	24
x.	ifEntry	36	24
xi.	ipNetToMediaEntry	36	23
xii.	udpEntry	36	23

Table 6.4: (continued).

OBJECT CLASS OR CATEGORY		$MHS_{full}(Kbytes)$	$MHS_{mod}(Kbytes)$
4.	Retrieving an attribute each from different object classes		
i.	interfaces2	36	25
ii.	ip2	36	27
iii.	udp2	36	24
iv.	tcp2	36	25
v.	system3	45	36
5.	Retrieving an entire column of a tabular object		
i.	atEntry	36	22
ii.	tcpConnEntry	36	23
iii.	udpEntry	36	22
iv.	ipNetToMediaEntry	36	22
v.	ipAddrEntry	36	22
vi.	ipRouteEntry	36	23
vii.	ifEntry	36	23
6.	Retrieving an entire table		
i.	atEntry	36	23
ii.	tcpConnEntry	36	24
iii.	udpEntry	36	23
iv.	ipNetToMediaEntry	36	23
v.	ipAddrEntry	36	23
vi.	ipRouteEntry	36	25
vii.	ifEntry	37	29
7.	Setting a single attribute		
i.	ifAdminStatus	36	25
ii.	ipForwarding	36	24
8.	Setting a row entry		
i.	ipNetToMediaEntry	36	25
ii.	ipRouteEntry	36	26
9.	Creating an instance		
i.	ipNetToMediaEntry	36	25
ii.	ipRouteEntry	36	27
10.	Deleting an instance		
i.	ipNetToMediaEntry	36	23
ii.	ipRouteEntry	36	23
11.	An event report		
i.		36	23

indicates the maximum heap size for individual queries. The MHS_{mod} column indicates modified measurements of the maximum heap size obtained by ignoring the allocations of the 8200 bytes required by functions that interact with the file system, like `fgets()` and `fprintf()`. These functions represent incidental allocations and are not directly related to the management protocols. In all cases the query was tried with different instances of the object class in question, or with different attributes. Varying the object class or the attribute usually leads to a variation in the observed MHS value. These variations do not appear in the tables due to round off and will not be considered further.

Since the MHS_{full} value remains largely constant in Table 6.4, it is clear that Figure 6.2 (top) is indicative of the memory allocation/deallocation pattern of the CMIP agent. In the overwhelming majority of cases, the allocation of 8200 bytes masks other allocations. Hence no correlation can be drawn between the MHS_{full} value and the query used. The MHS_{mod} values however provide a clearer picture, since the influence of incidental allocations has been removed from these observations. Further analysis of the CMIP agent will therefore be based on this column of data.

While retrieving a complete object instance (Table 6.4, item 1), the MHS_{mod} values vary directly with the number of attributes that are present in the object class being retrieved. The `icmp` (26 attributes) and `ifEntry` (23 attribute) object classes have the largest MHS_{mod} values, whereas the `interfaces` (2 attributes) object class attains the lowest value. The other attributes fall in between, roughly in accordance with the number of attributes they contain. This suggests that this pattern is largely due to the memory required to encode the response PDU, since in a CMIP agent, that is the only aspect that would vary with the number of attributes in the target object class.

Item 2 uses scoping to retrieve an instance each from two different object classes. It may be observed that the MHS_{mod} values are greater than those for any of the constituent object classes (as observed in item 1). This difference rises with the number of objects in the scoped subtree, being the minimum for the `udp2` category which scopes the MIB subtree rooted at the `udp` object class, and maximum for the `system3` category which scopes the MIB subtree rooted at the `system` object class. This indicates that there is a *scoping overhead* similar to what was observed in the performance studies. The exact nature of this overhead can be determined by checkpointing the agent and stepping through relevant sections of the code if necessary. This overhead is also apparent in item 4 where scoping is used to extract individual attributes from two separate object classes.

Item 3 deals with retrieving single attributes from an object class. No noticeable

trend exists here, though there is a slight, almost imperceptible variation with the total number of attributes in the containing object class. This is due to the fact that the current implementation extracts the values of all the attributes before returning the target attribute. This variation may be eliminated by extracting only the desired attributes.

The MHS_{mod} values obtained during the retrieval of an entire column of data from a tabular object (item 5) show no apparent trend. Though one might have expected some dependency on the size of the table, that is not the case, since the entire table is never required to be in the agent's process space. Each row is read in from system space (the kernel memory in this case), the value of the target attribute is then sent out to the manager in a separate linked reply PDU, which is then freed.

While retrieving entire tables, however, there is a dependency on the number of attributes in the object being retrieved, i.e., the number of columns per row of the table. This is because the values of all the attributes in a row are part of a linked reply PDU. This variation is clear with `udpEntry` (3 attributes) requiring an MHS_{mod} of 23 Kbytes, whereas `ifEntry` (23 attributes) requires an MHS_{mod} of 29 Kbytes. This is due to the additional dynamic memory required to encode all the attributes (columns) in a row prior to transmission to the manager.

The data in the remaining items of Table 6.4 show no significant trends, partly because there are not sufficient cases in each item for a trend to show up. However, it may be noted that in item 7, setting `ifAdminstatus` requires a greater amount of dynamic memory than setting `ipForwarding`. In items 8 and 9, the operations dealing with `ipRouteEntry` involve a higher heap size than the corresponding operations on `ipNetToMediaEntry`. A more detailed analysis of the memory allocation/deallocation pattern will uncover the reasons for such behavior.

The Heap Size at Idle State

Table 6.5 presents the CMIP agent's heap requirements while it is idle and is waiting for an incoming query. These readings were obtained by observing the agent's memory allocation trace produced by the instrumented `mymalloc()` function.

Table 6.5: CMIP heap requirements (Kbytes) at idle state.

$HEAP_{full}$	$HEAP_{mod}$	MHS_{full}	MHS_{mod}
29	21	36	21

$HEAP_{full}$ represents the amount of dynamic memory actually held by the process just before it receives a query. $HEAP_{mod}$ is similar, except that the effects of incidental allocations due to file system I/O have been ignored in its calculation.

The values of the actual heap space held by the agent at idle time is important since this is a constant overhead imposed on the managed node. This is especially true if the node is not capable of swapping out the process during its idle periods.

The values of MHS_{full} and MHS_{mod} just before the agent receives a query show clearly that most of an agent's dynamic memory requirements are allocated at start up, even before a query has been processed. The following section will explain this.

Further Analysis of CMIP's Heap Requirements

Figure 6.3 shows the increase in MHS_{mod} values during the execution of the CMIP agent. The query retrieved all the attributes of the `icmp` class (26 attributes) and those for the `interfaces` class (2 attributes). The MHS_{mod} values for the latter query are shown in parenthesis. The data has been rounded off to the nearest Kbyte. It should be noted that the increase in the MHS_{mod} value during any stage of operation may be equal to or less than the actual number of bytes allocated during this stage. Nevertheless, the MHS_{mod} values provide sufficient insight in a succinct form into the memory allocation pattern of the agent.

At idle state the CMIP agent had a MHS_{mod} of 21 Kbytes. Checkpointing was used to partition this allocation into constituent parts. This will enable the evaluator to determine which parts of the protocol are memory intensive.

With the CMIP agent, setting up the initial environment by using the ISODE function `isodetailor()` and obtaining the application entity information (`aei`) moves up the MHS_{mod} to 7% of its final idle state value. Another ISODE function, `RyDispatch()` is used to register all the operations that may be invoked on the agent via the ISO remote operations facility. During agent initialization, setting up an in-core image of the MIB and support for print routines is responsible for the greatest increase in MHS size, as recorded in Figure 6.3. The latter may be removed since the function of these print routines is purely diagnostic.

The `AcInit()` function of ISODE is equivalent to the `A-ASSOCIATE.INDICATION` primitive on successful return. It runs up the MHS_{mod} to 96% of its final idle value. Finally, the ISODE function `AcAssocResponse()`, which is equivalent to the `A-ASSOCIATE.RESPONSE` on successful return, takes the MHS_{mod} to its final idle state value.

Once a query is passed to the agent, the agent first decodes the raw BER encoded

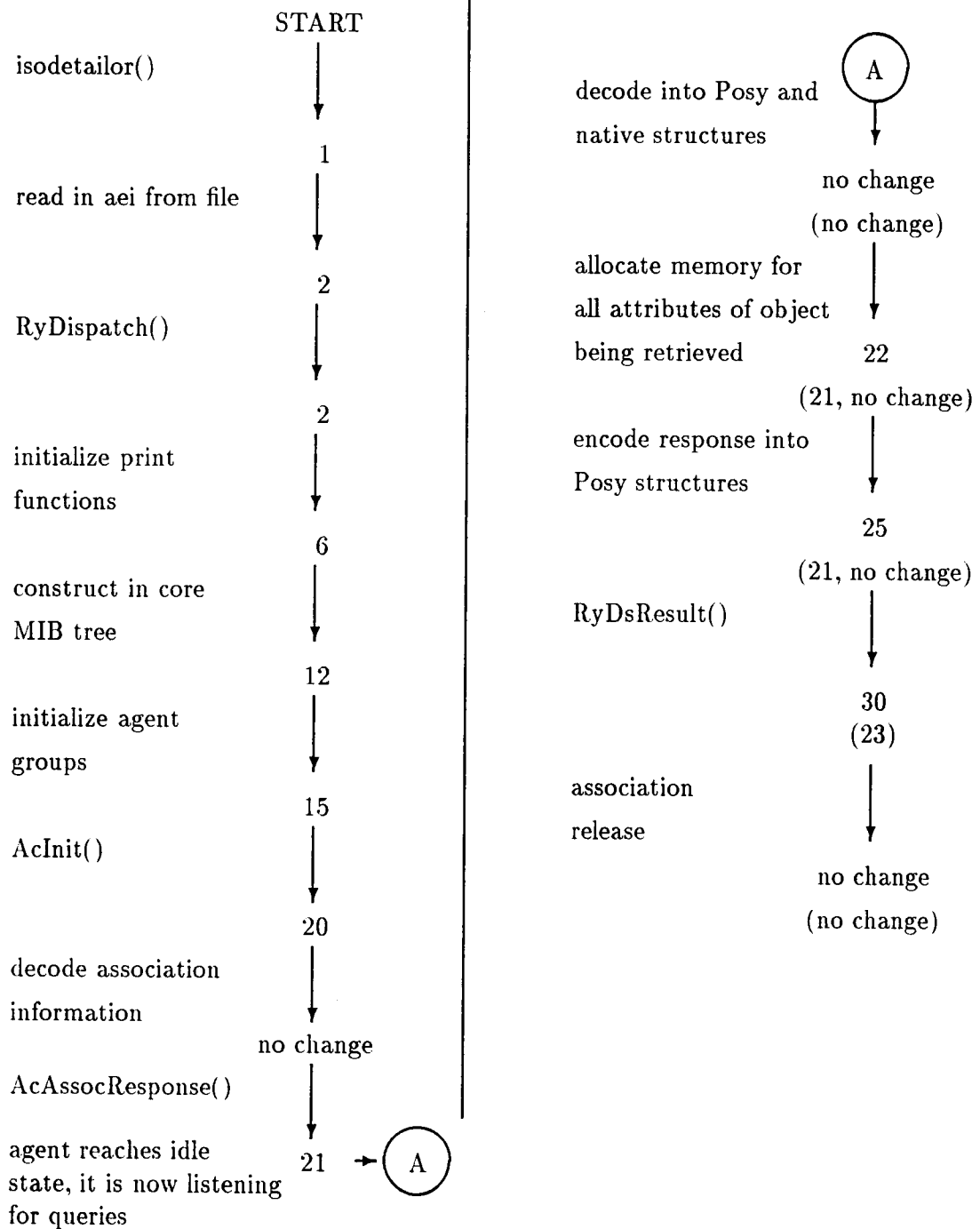


Figure 6.3: Increase in the MHS_{mod} values during the execution of a CMIP agent.

octets into the Posy generated intermediate structures. The dynamic memory required in this step depends on the complexity of the query, but it is unlikely that these allocations will cause the MHS to exceed the value already attained at this time. A relatively small amount (usually under 500 bytes, the exact amount depends on the complexity of the query) of heap space is required to convert the Posy generated structures to a native form.

As explained later, the agent puts aside memory for all the attributes of an object class, when it encounters any object class for the first time since start-up. This can be quite large for an object with many attributes like `icmp`. The heap space required to process the query is insignificant compared to the MHS. Encoding the response is however memory intensive. It varies with the number of attributes being returned in each response PDU. Since memory is freed after each PDU is transmitted, the number of PDUs in the response does not increase the heap size. Consider for example, the retrieval of all the attributes of a single object instance: encoding the 26 attributes of the `icmp` object class into the Posy generated structures causes the MHS_{mod} value to increase by 3 Kbytes. For the `interfaces` class with two attributes, there was no change to the MHS_{mod} value, though the amount of dynamic memory actually allocated to the process at that time increased by 510 bytes.

The ISODE function `RyDsResult()` converts the intermediate structures to BER encoded octets and invokes the lower layers to effect transmission of the response PDU. While retrieving all the attributes of the `icmp` class, this takes the MHS_{mod} value higher by 5 Kbytes to its final value of 30 Kbytes. For the `interfaces` object class, this increase was by 2 Kbytes, taking the MHS_{mod} to its final value of 23 Kbytes.

It should be noted that the purpose of providing these figures is not the precise measurement of the agents used in the experiments. The aim is to show what aspects of the protocol and the networking environment (ISO over TCP, or pure TCP) are memory intensive. This will enable developers of new protocols and implementations to pay special attention to these areas.

The scoping overhead mentioned previously is due to the fact that when the agent encounters an object class for the first time, i.e. while processing the first query that causes it to encounter the object class on the MIB tree, it requisitions memory to hold details about all the attributes of that class in a convenient form. Hence when a scoping operation causes the agent to traverse a major part of the MIB tree, comprising a number of object classes, additional heap space is used up for storing the attributes of all those classes, even though the accompanying filter might not match it eventually. This allocation is retained throughout the lifetime of the agent. Hence when the same oper-

ation is performed the next time, the scoping overhead will not register on the memory allocation trace. This overhead is therefore entirely a quirk of the implementation. It is quite possible that a more carefully designed agent can eliminate this overhead.

Filtering requires a negligible amount of memory. In these experiments, about 20 bytes were required for each instance of an object class that passed through the filter, and this too only for instances of object classes whose list of contained attributes included the attributes mentioned in the filter. The fact that a number of linked replies are usually needed to respond to a scoped request does not affect the heap size adversely, since after each linked reply PDU is sent, the agent frees the corresponding heap space.

While examining the differences in heap sizes involved in setting the *ifAdminStatus* and *ipForwarding* attributes one notices that the difference is essentially because the *ifEntry* object class is deeper in the MIB tree than the *ip* class. It takes more memory to specify the full distinguished name of this class (which contains the *ifAdminStatus* attribute). This causes the ISODE *ros_work()* routine to allocate about 400 more bytes while decoding the BER encoded request to set the *ifAdminStatus* attribute. The rest of the difference is due to other implementation dependent factors and is of no real interest here. Differences in the dynamic memory used while decoding the query is also partially responsible for the difference in heap size that was noticed while setting the *ipNetToMediaEntry* and *ipRouteEntry* entries (item 8, table 5.4). With reference to the CREATE operations involving the *ipNetToMediaEntry* and *ipRouteEntry* object classes, this operation returns all the attributes values of the newly created object. It takes more dynamic memory to encode the results for *ipRouteEntry* (12 attributes) than it does for *ipNetToMediaEntry* (5 attributes). The *MHS_{mod}* value of the former outstrips the latter by about 1 KB at this stage.

The Heap Segment of the SNMP Agent

The Maximum Heap Size for Individual Queries

Table 6.6 lists the maximum heap sizes for individual queries. The column headings have the same meanings as those in table 5.4, and are explained at the beginning of Section 6.4.3.

Figure 6.2 (bottom) is indicative of the memory allocation/deallocation pattern of the SNMP agent. Therefore there is no difference between the *MHS_{full}* and *MHS_{mod}* columns.

Close examination of the data in section 1 of Table 6.6, retrieving a complete

Table 6.6: SNMP agent heap sizes.

OBJECT CLASS OR CATEGORY		$MHS_{full}(Kbytes)$	$MHS_{mod}(Kbytes)$
1.	Retrieving a complete object class		
i.	atEntry	58	58
ii.	icmp	60	60
iii.	ipAddrEntry	53	53
iv.	interfaces	51	51
v.	ipRouteEntry	57	57
vi.	ip	58	58
vii.	udp	52	52
viii.	tcp	64	64
ix.	tcpConnEntry	61	61
x.	ifEntry	59	59
xi.	ipNetToMediaEntry	59	59
xii.	udpEntry	60	60
2.	Retrieving an instance each from different object classes		
i.	interfaces2	60	60
ii.	ip2	64	64
iii.	udp2	61	61
iv.	tcp2	66	66
v.	system3	67	67
3.	Retrieving a single attribute from an object class		
i.	atEntry	57	57
ii.	icmp	51	51
iii.	ipAddrEntry	51	51
iv.	interfaces	51	51
v.	ipRouteEntry	52	52
vi.	ip	51	51
vii.	udp	51	51
viii.	tcp	51 – 60	51 – 60
ix.	tcpConnEntry	60	60
x.	ifEntry	51	51
xi.	ipNetToMediaEntry	57	57
xii.	udpEntry	59	59

Table 6.6: (continued).

OBJECT CLASS OR CATEGORY		$MHS_{full}(Kbytes)$	$MHS_{mod}(Kbytes)$
4.	Retrieving an attribute each from different object classes		
i.	interfaces2	52	52
ii.	ip2	58	58
iii.	udp2	59	59
iv.	tcp2	60	60
v.	system3	61	61
5.	Retrieving an entire column of a tabular object		
i.	atEntry	57	57
ii.	tcpConnEntry	60	60
iii.	udpEntry	59	59
iv.	ipNetToMediaEntry	57	57
v.	ipAddrEntry	51 – 52	51 – 52
vi.	ipRouteEntry	52	52
vii.	ifEntry	51 – 57	51 – 57
6.	Retrieving an entire table		
i.	atEntry	58	58
ii.	tcpConnEntry	61	61
iii.	udpEntry	60	60
iv.	ipNetToMediaEntry	58	58
v.	ipAddrEntry	54	54
vi.	ipRouteEntry	62	62
vii.	ifEntry	65	65
7.	Setting a single attribute		
i.	ifAdminStatus	51	51
ii.	ipForwarding	51	51
8.	Setting a row entry		
i.	ipNetToMediaEntry	52	52
ii.	ipRouteEntry	52	52
9.	Creating an instance		
i.	ipNetToMediaEntry	52	52
ii.	ipRouteEntry	52	52
10.	Deleting an instance		
i.	ipNetToMediaEntry	51	51
ii.	ipRouteEntry	52	52
11.	An event report		
i.		52	52

instance(i.e., all the attributes) of an object class, shows the influence of two factors — the number of attributes being recovered and the size of the associated table. For *icmp*, *ipAddrEntry*, *interfaces*, *ip*, *udp* and *ifEntry* where no tables or very short tables are involved, the MHS varies with the number of attributes recovered. This pattern is lost only when the object class requires a relatively large table to be read in (i.e., *atEntry*, *tcpConnEntry*, *udpEntry*), when the size of this table also influences the MHS value, since the agent in use recovers the entire table regardless of how many instances of the class are actually requested by the agent. This saves time if the next few queries also reference items that belong to the same table, for then the table will not have to be read in from system space once more.

In item 2, an instance each is recovered from two different object classes. SNMP does not use scoping. To achieve this effect, the query must specifically identify all the attributes of the two instances of the object classes that are to be recovered. Hence one cannot expect the heap requirement to be the maximum of the two constituent classes (and therefore derivable from item 1). Rather, it should be greater than either of them, since a greater number of attributes are to be processed with a single request PDU and a single response PDU. Hence no memory can be released after the queries belonging to a particular object class have been processed (as was possible with CMIP). As a matter of fact, the SNMP protocol does not recognize the concept of object classes.

Retrieving a single attribute (item 3), is clearly varying with the size of the tables involved. Note that the *tcp* object class stands out because of the wide range of MHS values it exhibits: 51 – 60 Kbytes. This happens because recovering the *tcpCurrEstab* attribute requires the agent to extract the entire *tcp* connection table before searching through it for established connections. This is the only attribute that produces a high MHS value. The MHS values for the other attributes were equal or close to the lower value of 51 Kbytes.

While retrieving an attribute each from two different object classes (item 4), the MHS is greater than the MHS values for recovering the individual attributes (as observed in item 3). The increase is nominal unless both attributes have large tables associated with them. This would cause the MHS to rise further. While retrieving an entire column of a tabular object, the MHS clearly varies with the size of the table. The values for *ifEntry* are of special interest because of the wide range: 51 – 57 Kbytes. The upper value represents the last attribute under the *ifEntry* subtree, so while processing the last GET-NEXT operation on this attribute, the agent goes over to the next item on the MIB tree, the address translation table. Retrieving this table causes the MHS value to increase drastically.

For retrieving a full table, the manager uses successive GET-NEXT requests. The MHS varies with the table size and the number of attributes in each row. The MHS values obtained here should be similar to those observed in item 1. That is indeed the case except for the `ipRouteEntry` and `ifEntry` object classes. These objects are followed by the `ipNetToMediaEntry` and `atEntry` objects respectively. Since the final GET-NEXT causes these tables (the ARP table in either case) to be recovered, the MHS value also increases. *The lesson to be learned is that alternating large tabular objects with easily retrievable objects in the lexicographic ordering will decrease the average host impact due to GET-NEXT operations on an SNMP agent, or any agent that follows a similar protocol.* This is because if two large tabular objects are successively ordered, then recovering the first one with the GET-NEXT will also involve retrieval of the subsequent large object. This would not be as detrimental if the second object had been an inexpensively retrievable object.

Setting objects (which also includes creating and deleting objects) does not involve extracting entire tables. The MHS values in items 7, 8, 9, and 10 are therefore essentially similar. There is some difference between 7i, 7ii, and 10i with the others in the group since the requests are much shorter in length and involve only a single attribute. Therefore there is less decoding and encoding involved.

The Heap Size at Idle State

Table 6.7 presents the SNMP agent's heap requirements while it is idle and is waiting for an incoming query. These readings were obtained by observing the agent's memory allocation trace produced by the instrumented `mymalloc()` function.

Table 6.7: SNMP heap requirements (Kbytes) at idle state.

$HEAP_{full}$	$HEAP_{mod}$	MHS_{full}	MHS_{mod}
41	41	45	41

The significance of these data has been explained in the corresponding section on CMIP.

Further Analysis of SNMP's Heap Requirements

Figure 6.4 shows the increase in MHS_{mod} values during the execution of an SNMP agent. The query retrieved all the attributes of the `icmp` class (25 attributes) and those for the `interfaces` class (1 attribute). The MHS_{mod} values for the latter query

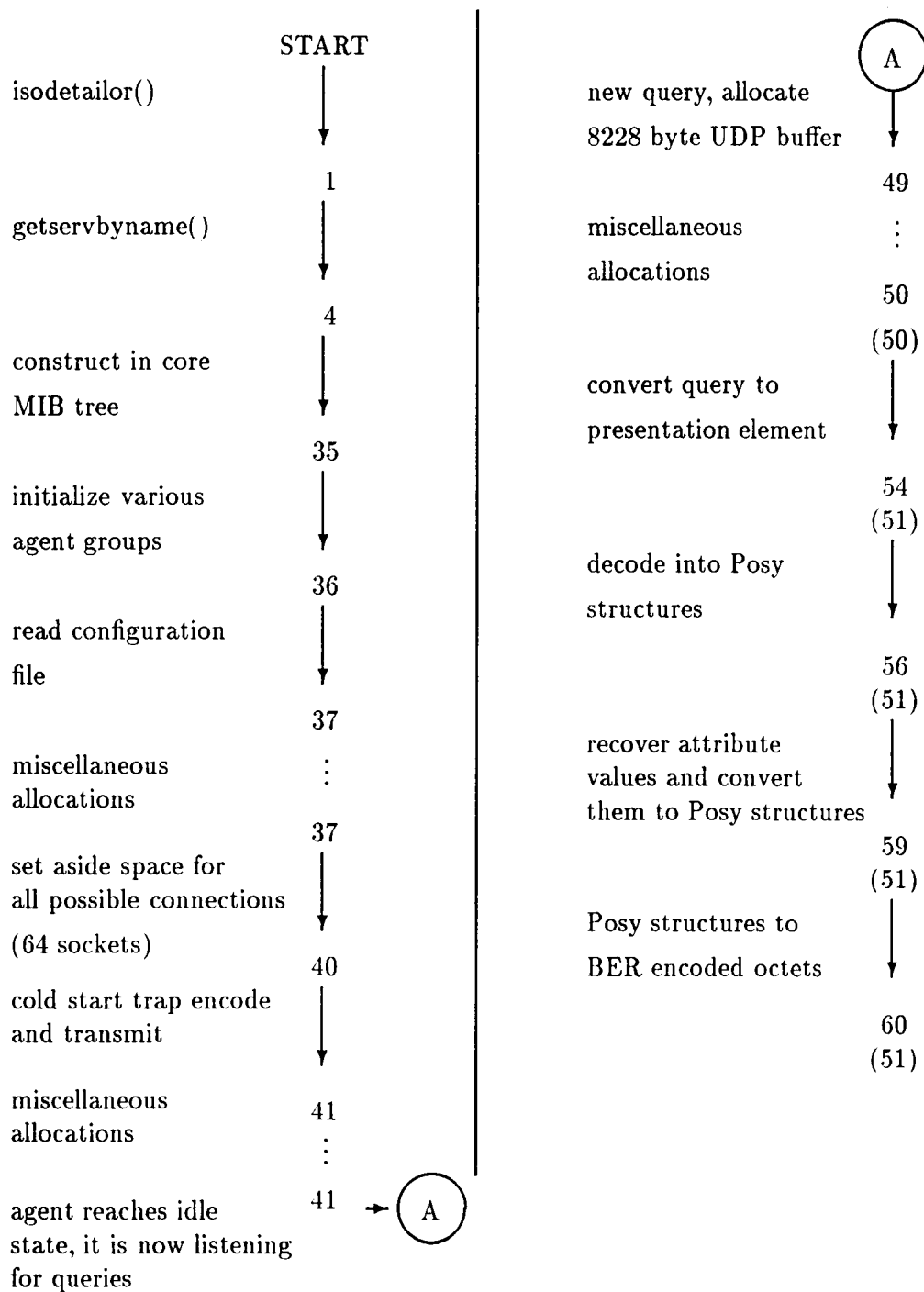


Figure 6.4: Increase in the MHS_{mod} values during the execution of an SNMP agent.

are shown in parenthesis. The data has been rounded off to the nearest Kbyte. Though the figure is largely self explanatory, some of the results merit separate discussion. The system function `getservbyname()` is used initially to check if the `snmp` service exists on the node. This use of memory is largely system specific, and hardly relevant to the protocol.

Constructing the in-core MIB tree increases the MHS_{mod} value to 35 Kbytes. This is the single most expensive stage of the agent's operation. Since a single agent process can handle multiple manager stations, it creates a table for all the possible connections it can handle. The current system permits a process to have 64 open sockets, and hence that many active connections. The allocation of a 8 KByte buffer to process the UDP datagrams causes the MHS_{mod} value to increase further.

Since the agent uses the ISODE transport layer functions to access UDP, it requires an additional conversion stage to convert data from a presentation stream to a presentation element. This matter is a consequence of using the ISODE generic transport layer interface, and could be avoided in other implementations that do not use ISODE. Presentation elements are fundamental to ISODE's ASN.1 facilities. They are essentially C structures that represent an ASN.1 value as a string of octets or bits (simple types) or as a linked list of other presentation elements (constructor types). Pepy generated routines are capable of converting these presentation elements to Posy generated structures and vice versa. The rest of the figure is easily understood.

Use of the ISODE transport layer interface to access UDP services and use of automatic encoding and decoding tools has resulted in the use of a large amount of dynamic memory. This could have been decreased with a different implementation strategy. Nevertheless, constructing the in-core MIB tree and encoding and decoding the management PDUs remain the most expensive parts of an agent with respect to its dynamic memory requirements. These stages therefore require careful consideration both at the protocol design phase and during the implementation phase.

6.4.4 Closing Remarks

It is apparent that the greatest amount of dynamic memory is required to construct the run time MIB tree and to decode and encode the management PDUs. The SNMP dynamic space requirement for constructing the MIB tree is larger than that for CMIP, this does not mean that the SNMP MIB design is inferior in this respect. It so happens that much of the CMIP's in-core MIB has been placed in the data segment. What was seen in this section was literally the tip of the 'MIB tree iceberg'. Even analytically,

the CMIP MIB should require greater space since the individual MIB objects are more complex and will also require links to build both the registration and the containment tree, whereas SNMP only requires a registration tree. It is imperative that the SMI specifications of MIB structures be kept simple, since there will usually be hundreds of such objects in any MIB. Any complexity in representing individual objects will therefore add up to create a staggering host impact. In future network management protocols, every effort should be made to avoid the need for a complete in-core MIB or engineer the SMI so as to make the MIB extremely simple. This may place a limit on the types of objects that may be recovered by the manager. At present, implementors should consider this phase of the implementation carefully, as it is desirable to have the most compact run time representation of the MIB without sacrificing performance. This matter is further touched upon in the discussion on the data segment requirements of the protocols under study.

Encoding and decoding the management PDUs is the next major consumer of dynamic memory. This is clearly a result of the semantic gap between native and standardized data representations. Although differences in MIB organization can lead to shorter or longer queries depending on the task at hand, it is unlikely that any MIB organization will lead to consistently short queries for all operations. It is far more practical to streamline the encoding and decoding process, and construct the agent so as to reduce the semantic gap.

6.5 The Data Segment

6.5.1 Measurement Strategy

The size of the data segment is computed by obtaining the symbol table of the agent via the Unix `nm` command and having the output sorted in the order of the symbol addresses by using the `-n` flag. No accurate determination can be made about the amount of data space required by different sections of the code since much of the data is declared as "static" within functions and therefore do not appear in the symbol table listing. Furthermore, much of the data is global and used by different stages of the protocol, making such a determination somewhat meaningless. Nevertheless, the symbol table approach does allow one to gain an understanding of the protocol's data requirement's. Total figures have been rounded off to the nearest Kbyte.

6.5.2 Data and Analysis

The Data Segment of the CMIP agent

The CMIP data segment has been partitioned into several components as presented in Table 6.8. The MIB component represents the data required to store the information

Table 6.8: The CMIP data segment.

agent component	size (in KBytes)
MIB	75.06
libisode	121.92
SMI (pepy generated)	11.10
ROS (pepy generated)	26.09
other	42.73
TOTAL	277

contained in the MIB. The extensive use of static memory has led to the use of less dynamic memory. The pros and cons of this trade-off are debatable. In general, the use of more dynamic memory will lead to a better use of available memory since it will only be allocated on demand, and released when no longer necessary. However, the additional system calls to allocate the memory will slow down the process. The situation while constructing the run time MIB tree is somewhat different since the memory is allocated when the agent starts up and is never freed during the lifetime of the agent. Hence the additional delay incurred to allocate the memory dynamically will not be of any consequence after the start up period. As a result it would be better to use dynamic memory to construct the run time MIB tree. This would also allow agents to create new MIB objects on the fly. A point to note is that read only data pages can be shared among different processes. Hence, if different processes, e.g., two different agents, have a common MIB, it would be economical to store it in a read only data area.

The libisode component of the data segment represents data space required by the ISODE library. This contains the association control, presentation, session, and transport layers. The remote operation services, the interface between the ISO TP0 and TCP, and the data conversion routines specified in item 12 of Table 6.2 are also part of this library. Pepy automatically generates functions to print, encode, decode, and free the ASN.1 structures specified in the SMI and in the RO-notation specification of CMIP. The data area required by these functions amounts to 11.10 KBytes and 26.09

KBytes respectively as shown in Table 6.8. Everything else has been grouped under “other”. This group includes the data associated with components 3 to 4b, 7 to 11g, and 14 to 16 in Table 6.2.

The Data Segment of the SNMP agent

The SNMP data segment has been divided into several components and presented in Table 6.9.

Table 6.9: The SNMP data segment.

agent component	size (in KBytes)
MIB	none
libisode	32.09
SMI (pepy generated)	see text below
other	126.55
TOTAL	159

The run time MIB tree has been implemented completely as dynamic memory, hence no part of the data segment has been used for this purpose. Since SNMP does not use ROS, or the association control, presentation, and session layers, only a small part of the ISODE facilities have been required. This is seen in the small size of the libisode component. SNMP does use the ISODE data conversion routines, ASN.1 facilities and the ISO connectionless transport facility to access UDP. The data area required by the pepy generated code to print, free, encode, and decode the ASN.1 structures specified in the SNMP SMI is minimal (below 2.37 KBytes) and has been included as part of the “other” component in Table 6.9. Notice that the largest chunk of the data segment is under the “other” component. Interestingly enough, a large part of this (94 KBytes) is taken up by a single array of 256 structures. Each structure represents a transport address and is 376 bytes long. The agent is capable of handling multiple requests from different managers. This array of transport addresses is used to keep track of the different managers who have placed a management request. While working with one manager at a time it is quite sufficient to have only a single entry in this array, rather than the 256 used here. This means that the SNMP agent could be considerably smaller if the situation warrants it.

6.6 The Stack Segment

6.6.1 Measurement Strategy

The stack size change dynamically. It must be monitored continuously in order to determine its maximum size. The lowest sampling period on the Sparcstation 2 is 10 ms. Since this is a long period compared to the running times of many of the functions used by a typical management agent, the interrupts for sampling may occur just before or after stack area has been allocated, utilized, and freed. This will cause one to miss the peaks of the stack size profile. Figure 6.5 illustrates this hypothetical situation. In this figure, the vertical arrows mark the sampling intervals.

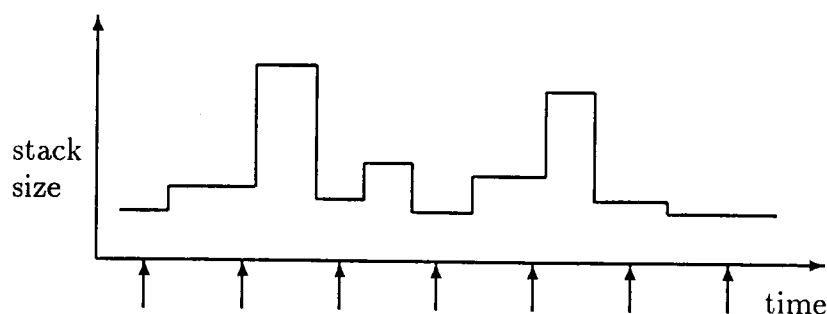


Figure 6.5: A situation where sampling occurs when the stack size is sub-maximum.

The following technique has been adopted to solve this problem:

Buried in the Unix kernel is a function called `sstk1()` in the file `kern_mman.c`. The function `sstk1()` modifies the stack size of every process as necessary and also changes its process table entry accordingly. This function was instrumented to print out the process id and stack size of every process that invoked it. The kernel was then rebuilt, thus allowing the experimenter to obtain a reading of a process' stack size as soon as it grows or shrinks.

6.6.2 Data and Analysis

Both the protocols under study required 20 KBytes of stack area. The small stack requirement of the agent is easily explained by the fact that data is put on the stack sequentially as different functions are invoked and immediately popped once the process exits the function. Since the level to which function calls are nested is limited, unless there is a significant amount of recursion, the stack never gets a chance to grow. Incidentally, the stack size reaches its maximum value during agent start up. There is

no further increase while it processes the manager's queries.

6.7 Research Question 2d: On CMIS's Functional Unit Groups

The question is reiterated here:

What are the host impacts of the different CMIS functional unit groups?

Section 1.4 introduced the different functional units of CMIS. The preceding investigation into the memory requirements of the different services and mechanisms provided by CMIS/CMIP will enable one to determine the relative impact of each functional unit. The CMIP text segment measures reported in Section 6.2.2 pertain to a controlled agent. Recall that a full agent has not been constructed since no standard actions have been defined as yet.

By studying the symbol table produced by the *nm* facility, it has been found that the size of the text required to move from a simple to a controlled agent is 79 KBytes for the implementation under study. This is due to the addition of the create and delete services, as well as the scoping and filtering mechanisms. Table 6.10 shows the additional memory requirements, partitioned into agent components. The indices of the agent components (left most column) are the same as those given in Section 6.3.1.

It is interesting to note that though filtering does not cause a great drop in the performance of the protocol when simple filters are used, the code *pepy* generates to encode, decode, free, and print them is considerable as shown by the entries in columns 4c, 4d, 5, and 6 under the column titled "SMI & MIB". As a matter of fact, the ASN.1 specification for the filter is required in both the SMI and the ROS, this increases the code size slightly unless an arrangement is made to use only one set of definitions, otherwise two sets of encoding, decoding, free, and print routines are created.

The heap requirements for the controlled agent will increase as the newly introduced scoping operation may require heap space in excess of what is required by operations that do not require scoping. This is clearly illustrated by the data in Table 6.3.

The data space necessary to upgrade a simple agent to a controlled agent will increase. This increase is due to the space required to store entries for the data symbols local to the functions that will process the ASN.1 constructs that relate to the create and delete operations in the RO specification of the protocol. The agent functions that will actually provide the create and delete services will similarly add to the data

Table 6.10: Additional text required to move from a simple to a controlled agent.

no.	agent component	size in Kbytes		
4a	encoding: native $\rightarrow$ intermediate	0.54		
4b	decoding: intermediate $\rightarrow$ native	1.78		
		SMI & MIB	RO module	common
4c	encoding: intermediate $\rightarrow$ octets	4.23	10.05	
4d	decoding: octets $\rightarrow$ intermediate	5.69	13.07	
5	free intermediate structures	2.05	3.56	
6	display intermediate structures	4.96	11.29	
8	display native structures	8.02		
10	filtering (limited complexity)	1.80		
11	the management operations			
d, e	CREATE and DELETE	11.40		
TOTAL		79		

segment. It is difficult to quantify this space requirement as names of local symbols do not appear in the listing produced by *nm*, though they do take up address space. Experience however suggests that this overhead is marginal and should not influence the decision to move from a simple to a controlled agent.

In terms of performance, the simple agent will perform better than a controlled agent since the expensive scoping operations are not permitted on a simple agent.

The additional text required to move from a monitored agent to a simple agent is reported in Table 6.11. This text is due to the introduction of the set service and the linked reply mechanism.

Table 6.3 shows that the heap space required to get entire tables or to set a managed object is less than what is required by the monitored agent to get managed objects. Therefore the heap requirements will not change, though the data segment will increase slightly due to reasons similar to those mentioned previously.

The increase in memory requirements while moving from an event sender to a monitored agent is the greatest since the additional RO-notation specifications for the get service as well as the entire MIB must be handled by the more powerful monitored agent. Table 6.12 tabulates the additional text required to move from an event sender to a monitored agent. The major increase is due to the additional burden of decoding incoming queries. The event sender agent does not have to handle any incoming traffic unless confirmed event reports are used. Unconfirmed event reports are used here.

Table 6.11: Additional text required to move from a monitored to a simple agent.

no.	agent component	size in Kbytes		
		SMI & MIB	RO module	common
4c	encoding: intermediate → octets		6.10	
4d	decoding: octets → intermediate		7.91	
5	free intermediate structures		2.24	
6	display intermediate structures		7.50	
8	display native structures		1.17	
11	the management operations			
c	SET		8.17	
21	others		0.66	
TOTAL			34	

Table 6.12: Additional text required to upgrade from an event sender to a monitored agent.

no.	agent component	size in Kbytes		
		SMI & MIB	RO module	common
3	agent start up		3.34	
4a	encoding: native → intermediate		4.02	
4b	decoding: intermediate → native		7.98	
4c	encoding: intermediate → octets	3.00	14.84	
4d	decoding: octets → intermediate	22.84	26.91	
5	free intermediate structures	0.25	4.89	
6	display intermediate structures	7.50	19.49	
8	display native structures		11.80	
11	the management operations			
a	GET		19.17	
g	supporting functions		1.94	
14	MIB initialization		10.21	
15	MIB traversal		1.78	
16	agent group initialization		2.93	
TOTAL			165	

Note that most of the MIB is not required in an event sender agent. Hence most of the 75.06 KBytes of data area dedicated to storing MIB related information will be freed, except for what is necessary to store the information related to the notifications (event reports) defined in the MIB.

There will also be major reductions in heap space requirements because the in-core MIB tree is not necessary for an event sender agent. Neither is it necessary to initialize the agent groups since the agent will not need to recognize separate agent groups, unless the nature of the events dictate otherwise. In addition, the information content of an event report is low, hence only a small amount of dynamic memory is required to encode it.

Some additional code is necessary in an event sender agent in order to enable a manager to specify itself as a recipient of event reports. The CMIP agent used here forwards the event report to the destination with which an association already exists. However, a special managed object called **Event Forwarding Discriminator (EFD)** exists to specify a event report report recipient, as well as other information for the event reporting service. The manager is able to set this information in order to change the destination of the event reports. Including such a facility in a event sender agent would require the agent to support the set operation. This in turn would transgress the boundaries of the different agent functional unit groups. However, in order to provide full functionality, a commercial event sender agent would certainly have to permit limited set operations. The memory requirements of the agent would increase correspondingly.

The decision to upgrade an agent will depend on the available memory and CPU cycles that can be dedicated to the task of network management. Moving from an event reporting agent to a monitored agent, or from a simple to a controlled agent requires a considerable amount of investment in terms of memory and CPU time. However, upgrading from a monitored to a simple agent is relatively painless in terms of the additional resources that are necessary. Administrative policy will therefore play a larger part in deciding on this upgrade. Referring back to Table 1.1 shows that the services provided by SNMP lie somewhere between the services provided by the simple agent and those provided by the controlled agent. Hence to obtain the functionality of SNMP, the controlled agent must be employed.

6.8 Conclusion

If the text section is considered, perhaps the greatest impact of protocols that use the OSI protocol stack is due to the use of the ROS facility. Therefore the ROS specifications of a protocol should either be kept simple or the corresponding code should be more compact. The code for encoding and decoding data and the routines necessary to convert data from one form to another can also be quite sizable unless special care is taken during protocol design and its implementation.

As far as the heap size is concerned, constructing the in-core MIB tree and encoding and decoding the PDUs provide the greatest host impact. It is therefore recommended that the greatest consideration be given to specifying the SMI and MIB during protocol design and later to developing a MIB tree implementation that strikes a balance between memory usage and performance. For connection-oriented protocols such as CMIP, setting up the initial connection requires a large amount of dynamic memory. Space must be allocated for the storage of information about the association and the user data that accompanies the request for association establishment.

The data size for CMIP turns out to be larger, especially since much of the run time MIB tree has been placed in non-dynamic memory, as data. This brings up the question of the advisability of using data space instead of heap space. The use of dynamic memory is preferred, unless the MIB is to be shared between different processes.

The combined text and data requirements for the locally available implementation of CMIP is 912 Kbytes, whereas the combined heap and stack requirements are 50 Kbytes. For SNMP, these figures are 391 and 80 KBytes respectively. The CMIP implementation reduced its dynamic memory requirement by using more static storage.

The memory required to upgrade from one functional unit group to the next higher group has been specified. Upgrading from an event reporting agent to a monitored agent, or from a simple to a controlled agent will involve a marked increase in memory and CPU requirement. However, moving from a monitored to a simple agent is relatively painless in terms of the additional resources that are necessary.

It should be kept in mind that though using an implementation to evaluate a protocol has sound practical value and leads to many insights into the performance and functioning of the protocol, the fact remains that the evaluation should undergo constant revision in the light of new implementation strategies that are sure to arise as technology progresses. This is because the performance and host impact of a protocol is closely related to its implementation technique [69, 72].

Chapter 7

The Management System's Impact on the Network

7.1 Objectives and Methodology

Research question 3 deals with the management system's impact on the network. This question will be addressed by evaluating the management framework's impact on the network in terms of the following:

1. the number of packets generated by the system (the manager and the agent), and
2. the number of bytes that are transmitted by the system.

Each of the above divisions can be further subdivided as follows:

1. Association establishment.
2. The one-query-one-object (OQOO) style query, where the agent returns a single object, linked replies are not used.
3. The one-query-multiple-objects (OQMO) style query, where the agent may return more than one object using the linked reply or some other technique.
4. Association release.

All the categories may not be applicable to every network management protocol. These categories are in keeping with the basic principle that underlies the synthesis of the evaluation procedures (Section 1.3). This principle dictates that the facilities

provided by the management framework should be divided into groups that are likely to exhibit different characteristics with reference to the evaluation methods.

Based on experience with CMIP and SNMP, it may be assumed that all queries will fit into a single packet. A response sent to a OQOO query will also be contained in a single packet. Responses generated in reply to a OQMO query may require multiple packets for transmission, because the reply may contain multiple object instances.

The bandwidth taken up by the network management system is of primary interest only when the net is functioning normally, i.e., with no noticeable loss of packets. This is so, because when the net is facing a congestion collapse, network management traffic takes priority, and the amount of bandwidth it requires is of less concern.

The preceding discussion is applicable to all network management protocols that follow a query-response model. The network impact of CMIP and SNMP will allow us to apply and elaborate on the evaluation strategy just presented. All measurements are under conditions of zero packet loss.

7.2 The Number of Packets Generated by the System

7.2.1 Association Establishment

The association establishment phase is present for CMIP only. In Table 7.1, the first row presents the total number of packets and the total number of bytes required for the association establishment phase, including acknowledgements and packet headers. The three-way handshake required by TCP to establish a connection is not considered part of the traffic generated by the management protocol, rather, it is generated by the underlying network. The second row shows the number of packets and bytes generated by the manager and the agent themselves.

Table 7.1: The number of packets and bytes required by the association establishment phase of CMIP.

	packets	bytes
total	10	989
manager and agent only	4	412
54 bytes are added to each packet by TCP, IP, and the Ethernet driver		

7.2.2 One-Query-One-Object Interactions

One-query-one-object interactions are applicable to both CMIP and SNMP. The models discussed previously will be used to predict the number of packets required wherever possible, since that will eliminate the need to conduct experiments.

From the model for the data transfer phase of a connection-oriented protocol in Section 4.3.2, it may be shown that when $p = 1$ and $q = 0$, the expected number of packets required for n queries is

$$3.2n - 0.5 \approx 3.2n \quad (7.1)$$

Of this, the expected number of acknowledgement packets generated by the underlying transport protocol is

$$1.2n - 0.5 \approx 1.2n$$

From the analysis of the connectionless protocol in Section 4.4.1, we find that when $p = 1$, the expected number of packets required for n queries is

$$2n \quad (7.2)$$

No acknowledgement packets are generated by the transport protocol.

7.2.3 One-Query-Multiple-Objects Interactions

Only CMIP provides a true OQMO type of query. These are useful for the bulk retrieval of entire tables (i.e., routing tables). Since no model exists for this mode of interaction, experimental data has been obtained to assess its impact on the network. Table 7.2 provides these data. The IP routing table was retrieved in each case. Each item in the table has been averaged over 10 observations, which provides a confidence interval of 10% at a 90% confidence level.

SNMP does not have any OQMO type of queries, nevertheless it provides for the retrieval of entire tables with the GET-NEXT service. This involves $(n + 1)$ GET-NEXT operations to retrieve a table comprising n entries. Therefore from the analysis of Section 4.4.1, the number of packets required for this would be

$$2(n + 1) \quad (7.3)$$

No acknowledgement packets are generated by the underlying transport protocol.

Figure 7.1 graphs the number of packets required vs. the number of objects retrieved. The experimental data for the CMIP linked replies (OQMO) is very nearly linear and coincides almost exactly with the smoothed line that has been fitted to the data.

Table 7.2: The average number of packets required vs. the number of objects recovered using a OQMO type of query.

objects retrieved	packets generated by the management system	acknowledgement packets	total
5	3.20	1.30	4.50
10	5.60	3.10	8.70
15	7.20	4.10	11.30
20	9.00	4.60	13.60
25	11.00	5.90	16.90
30	12.90	7.00	19.90
35	15.00	8.30	23.30
40	17.10	9.50	26.60
45	18.50	10.40	28.90
50	20.50	11.30	31.80

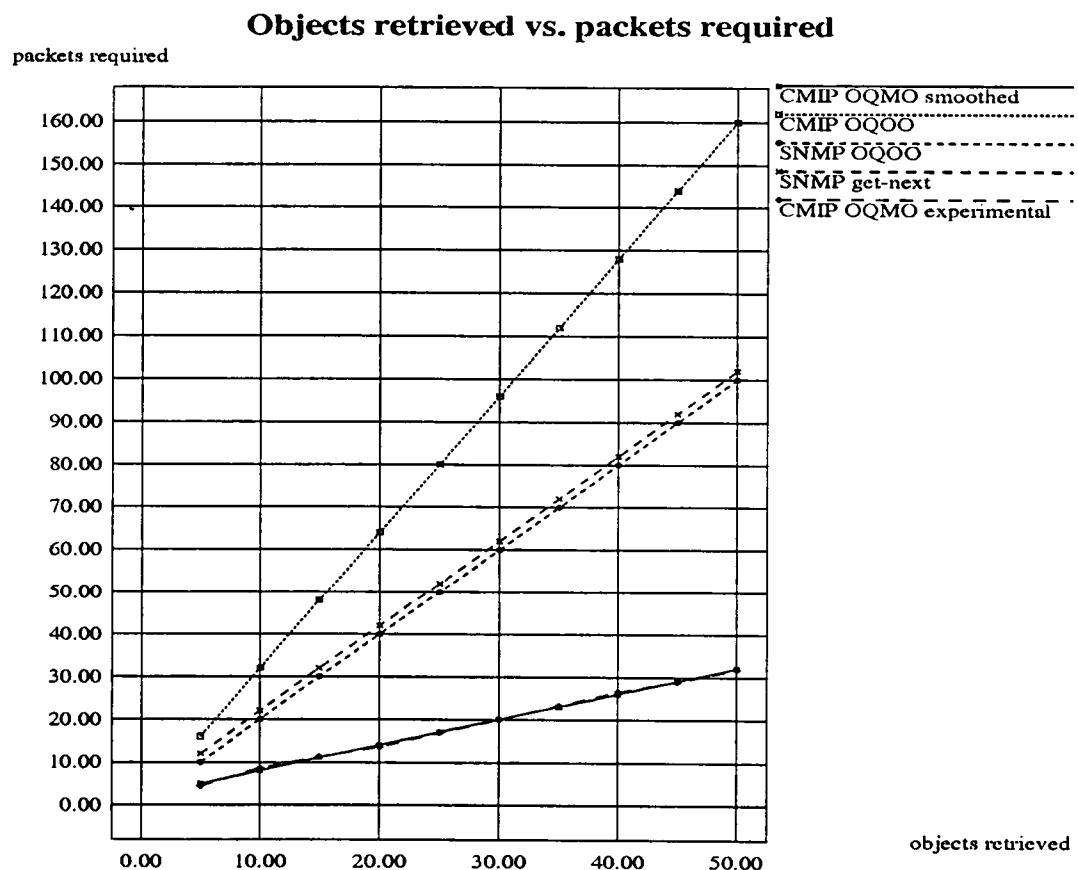


Figure 7.1: The number of objects retrieved vs. the total number of packets required.

The metric for the number of packets on the network can therefore be defined as *the number of packets required per object retrieved*. For the OQOO style of interaction, this quantity will be independent of the actual object retrieved, as long as the response fits into a single packet. For the OQMO style of interaction, the value of the metric will depend not only on the size of the reply associated with each retrieved object, but also on the time necessary to create successive responses. If the agent does not create successive linked replies quickly enough, the underlying transport protocol will not be able to place multiple replies in the same outgoing packet, a larger number of small packets will be created. In the worst case, each linked reply will cause a separate transmission. The `ipRouteEntry` table has been used here since it is simple to modify, and the size of a reply associated with each entry is of intermediate size. Table 7.3 presents the metric values for the various interactions. The number of packets required to establish and release the connection are not to be considered while calculating these values, as that would cause the metric value to vary inversely with the number of objects retrieved.

Table 7.3: The number of packets required to retrieve a managed object.

style of retrieval	packets per object retrieved
CMIP OQOO	3.2
CMIP OQMO	0.6
SNMP OQOO	2.0
SNMP with GET-NEXT	2.0

7.2.4 Association Release

This phase is present for CMIP only. Table 7.4 presents the total number of packets and the total number of bytes required for the association establishment phase. The first row presents the total number of packets and the total number of bytes required for the association release phase, including acknowledgements and packet headers. The connection release mechanism required by TCP is not considered part of the traffic generated by the management protocol, rather, it is generated by the underlying transport protocol. The second row shows the number of packets and bytes generated by the manager and the agent themselves.

Table 7.4: The number of packets and bytes required by the association release phase of CMIP.

	packets	bytes
total	7	466
manager and agent only	3	64
54 bytes are added to each packet by TCP, IP, and the Ethernet driver		

7.3 The Number of Bytes Generated by the System

The number of bytes used by CMIP for the association establishment and release phases have already been reported in Sections 7.2.1 and 7.2.4 respectively.

7.3.1 One-Query-One-Object Interactions

The number of bytes per query varies mainly with the number of attributes being requested and the number of attributes returned in the response. In protocols like CMIP, it also depends to some extent on the length of the distinguished name (DN) of the target object instance, the deeper it is in the MIB tree, the longer is its DN, therefore a greater number of bytes are required to encode it. With current MIBs this variation is constrained by the small depth of the MIB tree, besides, the CMIP standard states that an implementation can also use only a part of the distinguished name to identify an object instance. In the latter case, the variation of PDU size with the partial DN is lessened. There is no significant variation with the particular attribute being retrieved or its value. In this light, and in the interest of simplicity, the number of bytes generated by a network management protocol will be measured as a function of the number of attributes recovered only.

Figure 7.2 provides these data. Attributes of the `ifEntry` object class have been retrieved in order to gather the data. The number of bytes required for both CMIP and SNMP have been plotted. The total includes the acknowledgements (for CMIP only), and the packet headers on the data packets, as well as the actual management data generated by the manager and the agent. Each data packet transmitted by TCP requires 54 bytes of header information (including TCP, IP, and Ethernet headers), and each data packet transmitted by UDP required 42 bytes of header information (including UDP, IP, and Ethernet headers). The number of bytes generated by the manager and the agent themselves has been shown separately in Figure 7.2.

Findings

The main result is that the overhead generated by the underlying protocols for CMIP is greater than that for SNMP. This is because CMIP runs over a connection-oriented transport protocol which generates additional acknowledgement packets. Each data packet also has a packet header attached to it. The only overhead for SNMP are the packet headers added to each data packet.

The management information transmitted by CMIP requires a greater number of bytes than SNMP. This happens because the size of a CMIP query and response PDU is greater than the corresponding SNMP messages. Note that a CMIP query or response is transmitted wrapped in a ROS PDU, hence the increase in its size. If a full DN were used, that too would contribute significantly to the number of bytes required to encode

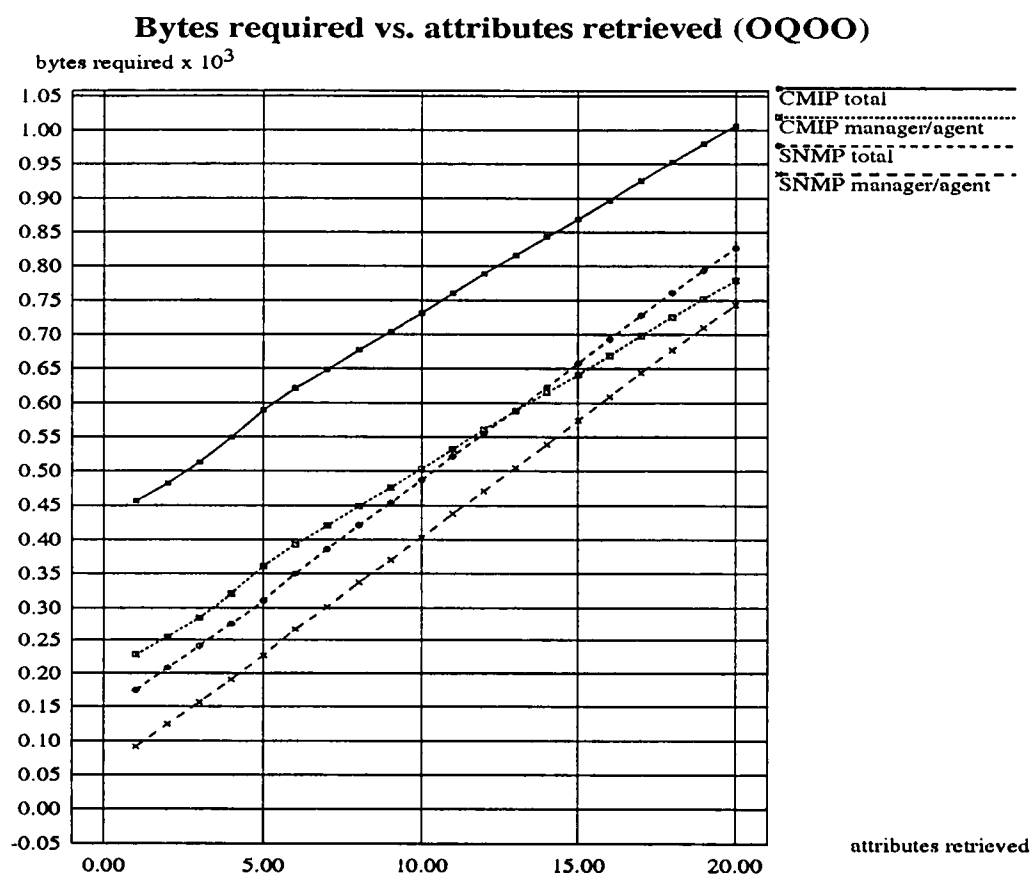


Figure 7.2: The number of bytes required vs. the number of attributes retrieved by a query.

a CMIP PDU. However, from the gradients of the CMIP and SNMP manager/agent lines, it is evident that the increase in the number of bytes per attribute recovered is greater for SNMP than it is for CMIP. The difference is small (6 bytes) and is probably due to the nested CHOICE constructs used by SNMP to encode the attribute values. CMIP uses the ANY DEFINED BY construct for this purpose. The net result is that when a greater number of attributes are recovered, the number of bytes generated by the SNMP manager and agent approaches that due to the CMIP manager and agent.

7.3.2 One-Query-Multiple-Objects Interactions

The number of bytes per query varies with the number of entries, i.e., the length of the table being retrieved and the number of attributes (columns) that have been requested. The length of the DN is also a factor as mentioned previously. However, to make a relative estimate of protocol behavior with regard to the number of bytes generated, it will be sufficient to measure the number of bytes as a function of the table length only, since that is the overriding factor. In this experiment the length of the `ipRoutingTable` was varied. A single attribute, `ipRouteDest` was recovered from each entry in the table. Figure 7.3 provides these data.

The number of bytes required for both CMIP and SNMP have been plotted. The total includes the acknowledgements (only for CMIP), and the packet headers on the data packets, and the actual management data generated by the manager and the agent. A data packet transmitted by TCP requires 54 bytes of header information, and each data packet transmitted by UDP required 42 bytes of header information. The number of bytes generated by the manager and the agent themselves has been graphed separately.

Findings

The following observations may be made from Figure 7.3 :

1. The overhead due to packet headers is larger for SNMP than it is for CMIP. This is due to the larger number of packets that SNMP uses. Since each packet has an overhead in the form of header information, the total overhead far exceeds that due to the few data packet headers and acknowledgements required by CMIP.
2. The total number of bytes used by CMIP is less than the total number of bytes used by SNMP. This too is due to the greater overhead incurred by SNMP.

3. The number of bytes of actual management information is less for SNMP. This means that each linked reply PDU generated by CMIP requires more bytes than the combined requirements of two SNMP messages (a query and a response), each containing a single attribute specification (i.e., a variable binding).

The curve for the total number of bytes required by CMIP is uneven since the number of acknowledgement packets generated is dependent more on the way the agent transport layer packetizes the output stream rather than on the length of the table being retrieved.

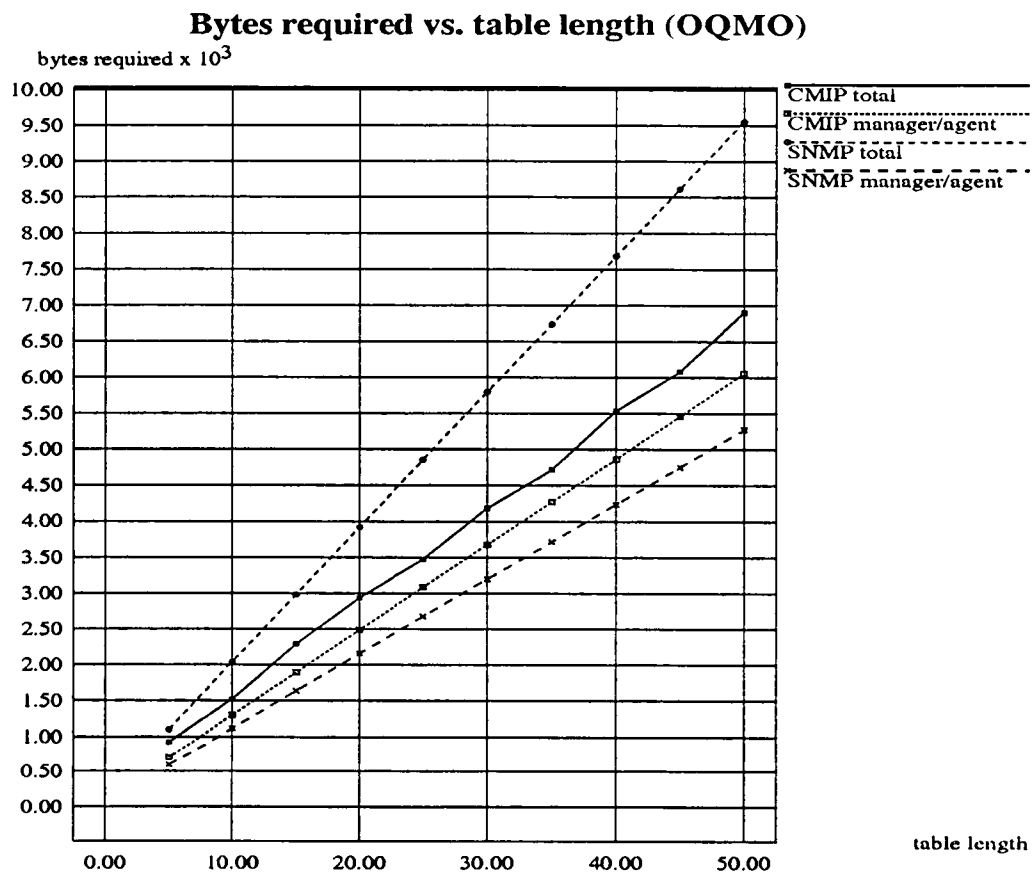


Figure 7.3: The number of bytes required vs. the length of the IP routing table.

7.4 Conclusions

The results of this chapter only serve to reinforce the point that neither a connection-oriented (CO) protocol, neither a connectionless (CL) protocol is best in all cases. We see that CO protocols generate lesser traffic during the retrieval of entire tables, whereas a CL protocol is more economical at other times. **A protocol that is capable of switching smoothly between CO and CL modes of operation depending on the job to be performed is therefore desirable.**

On a more specific note, Section 7.3.1 shows that the CMIP PDUs require a greater number of bytes. This is due to the ROS wrapper and the occasional need to encode the full distinguished name. It has been observed repeatedly, that though ROS facilitates the rapid prototyping of services, it is at a heavy price. Production implementations should therefore be able to bypass ROS in the interest of better all-around performance. Interoperability with ROS based implementations should not be sacrificed in the process. The need to encode a full DN stems from the fact that the same object class can be contained in multiple parent classes. Partial DNs may be used only in the absence of such a containment hierarchy.

Chapter 8

Summary, Conclusions, and Recommendations

8.1 Summary and Conclusions

This study has introduced a systematic methodology for the evaluation of network management protocols that follow a query response model. The methodology is explained in detail through its application to the Common Management Information Protocol (CMIP), and the Simple Network Management Protocol (SNMP) — two protocols that offer similar services, but are quite different in their approach to network management. Protocols which offer other services may also be evaluated within the same framework, though the basic operations will require redefinition in order to accommodate the new services. A comparative evaluation of two protocols is only possible to the extent they support common sets of management operations.

This section will summarize the evaluation strategy adopted to meet each objective of this dissertation. The application of these evaluation methods to CMIP and SNMP has led to a number of conclusions regarding the efficacy of these protocols. These conclusions are also highlighted.

8.1.1 Question 1: Evaluation of the Management Framework's Behavior in a Lossy Environment

This question deals with the evaluation of the management framework's behavior in a lossy environment.

Evaluation Strategy

The metrics here were the total expected number of packets and the time required to complete a management activity, given that each packet has a certain probability of being dropped. The connection-oriented (e.g., CMIP) and connectionless (e.g., SNMP) protocols do exhibit markedly different behavior. For low rates of packet loss, when connection failures will not occur, an absorbing Markov chain models the behavior of CMIP. The model assumes that each query and reply fits into a single packet, and a query is dispatched only when a response to the previous query has been received. For protocols operating in a connectionless mode, an analytic expression has been derived for determining the number of packets and the time required to complete an activity.

The situations that have not been modeled, i.e., the time requirements of CO protocols, CO protocols at a high rate of packet loss, and linked replies, have been examined experimentally. Experiments have also been used to verify the predictions of the newly introduced models. Since the experiments were conducted in an environment where the manager and the agent resided on adjacent machines on the same Ethernet section, the round-trip time may be assumed to be negligible. Hence, the times reported here provide an experimental lower bound on the expected times. A number of significant observations were made as a result of this investigation. They are summarized below.

Conclusions

The conclusions are:

1. When the rate of packet loss exceeds 50%, a significant number of connection resets occur during the association establishment phase. The number of connection resets increases rapidly at higher rates of loss. The same is true for the data transfer phase.
2. Data transfer with scoping and linked replies has two opposite forces governing the number of connections that will be lost during the retrieval of management information. The number of packets required to extract a number of managed objects is less than what it would be if individual queries were used to extract each item. This reduces the chances of a connection failure. On the other hand, as explained in the conclusion to section 4.3.5, the fact that the manager sends only one query at the beginning of the transaction also increases the chances of a connection failure or deadlock. As the number of objects to be retrieved increases, the second factor starts to predominate.

In CMIP, the only way of extracting entire tables is through the use of scoping and linked replies. Therefore the retrieval of large tables (with 50 or more entries) becomes virtually impossible under conditions of high packet loss ($> 50\%$) due to the large amount of time required. The "all or nothing" nature of linked replies aggravates this situation.

3. Connectionless protocols offer a graceful degradation of service over the entire range of packet loss rates. Therefore, at times of network collapse, such protocols are more effective than connection-oriented protocols.

8.1.2 Question 2a: Performance Evaluation of the Agent

This question deals with the performance evaluation of the agent.

Evaluation Strategy

It has been argued that a network management protocol consists of a set of services, i.e., GET, SET, etc. It is not a monolithic entity. Management of the network involves the use of a sequence of basic operations such as getting a single managed object, getting a full table, etc. These basic operations make use of the services proffered by the management protocol. Management operations other than the basic operations are possible, but they would be extensions or variations of one of the basic operations. The performance evaluation of the management protocol is therefore best done by assessing the performance the individual basic operations. A set of basic operations for CMIP and SNMP have been defined. The *gprof* profiling tool, available with many Unix systems, has been used to measure the performance of the basic operations. The resulting data has been analyzed with respect to the protocol's features. The major conclusions are presented next.

Conclusions

The conclusions are as follows:

1. Kernel reads are time consuming, especially if a large number of such reads are required to perform a given operation. This is because a kernel read requires a relatively expensive system call.
2. Scoping over a wide section of the MIB tree is an expensive process, as the agent must visit all the intervening objects in order to apply the filter to them.

3. In CMIP, the time spent in the underlying protocols is much greater than that for SNMP.
4. If no scoping is involved, the time spent in the CMIP agent itself is usually comparable to, or less than the time spent in the SNMP agent.
5. As far as performance is concerned, the greatest drawback of CMIP is due to the heavyweight underlying protocol structure as well as scoping across a large number of managed objects. The greatest promise is due to the fact that the object-oriented organization facilitates operations that involve a single object class only.
6. The performance (with respect to CMIP) of SNMP drops when multiple attributes of the same object class are requested in the same query, since a per attribute overhead is involved. Another drawback is the need to use multiple GET-NEXT operations in order to retrieve a full column or table.
7. The lexicographic ordering used by SNMP is not a burden, except that at times this SNMP implementation extracts a complete table and sorts it with the expectation that requests for other attribute values included in the same query will also require information from the table. In contrast, CMIP would extract only half the table on an average.
8. The simple filters used by this implementation of CMIP required a negligible amount of processing time.

8.1.3 Question 2b: Evaluation of the Agent's Memory Requirement

This question deals with evaluating the agent's memory requirements.

Evaluation Strategy

The host impact of a network management agent has been quantified in terms of the text, data, heap, and stack sizes. The text and data sizes remain constant during the execution of the agent. They have been obtained by utilizing the symbol table provided by the *nm* utility available on many Unix systems. When the symbol list is sorted in order of their addresses, it can be used to determine how much text is required by various components of the agent. It is not easy to determine which section of the code requires how much data, as much of it is declared as *static* in the various functions and

does not appear in the name list. The existence of global data, which is used by all the components of the agent, makes such a partition all the more difficult.

The heap and stack sizes vary as the agent executes. The functions that allocate heap and stack space have been instrumented to determine the maximum heap and stack sizes. The agent was checkpointed to determine the approximate amount of heap area required by the major sections of the code. Such a determination is not possible, nor is it necessary for the stack size. Since the heap requirements may vary with the operation being performed, the basic operations defined previously have been reused to obtain the heap requirements for the various basic operations. The conclusions drawn from this study are summarized below.

Conclusions

The conclusions are:

1. The remote operations support, application, presentation, and session layers constitute 29% of the CMIP text. This is not a problem on a powerful host. However, it may be a high overhead when one tries to introduce CMIP to resource feeble intermediate systems which would ordinarily need to support only the lower three layers of the protocol stack.
2. The code necessary to encode, decode, free, and display the structures specified in the RO specification occupies 22% of the total code. Clearly the RO notation is too complex. Fewer PDUs should have been specified. The SNMP approach (only two PDU types) is a good example. If SNMP were redesigned, it could be simplified further by using only a single type of PDU.
3. A drawback of the current data representation techniques is the amount of conversion necessary between the BER encoding and the native data representation that is usable by the agent. Bridging this "semantic gap" is necessary if implementations are to be compact.
4. Compared to SNMP, the SMI/MIB specifications of CMIP require greater amounts of code to process. These specifications should be streamlined with an eye on reducing coding requirements without sacrificing essential functionality.
5. The SNMP text size is smaller than the CMIP text size (232 KBytes against 635 KBytes). This is because SNMP does not require the upper layers of OSI, and since it provides for only two message types and has a simpler SMI, the amount of code necessary to process the relevant ASN.1 specifications is far less.

6. With reference to CMIP's heap requirements, the major contributions are due to the construction of a run time MIB tree, setting up the initial association, and the encoding of the responses generated by the agent for transmission back to the manager.
7. With SNMP, the greatest heap requirement is due to the construction of the run time MIB tree, and for allocating a 8 Kbyte UDP buffer.
8. The combined text and data requirements for the locally available implementation of CMIP is 912 Kbytes, whereas the combined heap and stack requirements are 50 Kbytes. For SNMP, these figures are 391 and 80 KBytes respectively. The CMIP implementation reduced its dynamic memory requirement by using more static storage.

8.1.4 Question 2c: What are the host impacts of the different functional unit groups of CMIP ?

This question deals with determining the host impacts of the different functional unit groups of CMIP. The four functional unit groups examined here are the event sender, monitored, simple, and controlled groups. The components of these groups have been presented in Table 1.1. As explained in Chapter 6, the symbol table, when arranged in order of increasing addresses, allows one to determine the text size due to the various components of the agent. By isolating the requirements of the create, delete, get, set, and event-report operations and those for the associated mechanisms like filtering, scoping, encoding, decoding, etc, it is possible to determine the amount of code required to upgrade from one functional group to the next. These results are summarized below.

move from	move to	additional code needed (KBytes)
event sender	monitored	165
monitored	simple	34
simple	controlled	79

This data will allow one to determine if the agent platform has sufficient memory to allow a more sophisticated functional unit. It should be noted that this data pertains to one particular implementation only, and could change if some other implementation is considered. The relative amount of code required to move from one functional unit to another is however expected to be similar. The execution times of the different

functional units can be gauged by the results of the performance evaluation presented in Chapter 5. From these results, one may determine if an agent may be upgraded to support additional functions.

8.1.5 Question 2d: What are the bottlenecks in CMIP and SNMP implementations?

This question deals with the aspects of CMIS and SNMP implementations that lead to poor performance. Figures 5.4 and 5.6 show that CMIP requires the greatest amount of time when scoping is performed over a wide section of the MIB tree. This high cost occurs because every object within the scoped subtree has to be examined to see if it matches the given filter. Such an operation is the only significant bottleneck in a CMIP implementation at this time. A strategy has been suggested to minimize the processing required to determine which objects in the scoped MIB tree match the given filter.

The results also indicate that the ISODE protocol layers below the application itself have a large processing time — these layers may prove to be the greatest bottleneck for CMIP or any other application that functions above ISO's seven layers.

It has also been observed that for some object classes, retrieval times increase because of the large number of kernel accesses required. This is true for platforms that keep management information in the kernel memory. The solution to this drawback would require the cooperation of the operating system itself, and is not in the hands of the protocol designer or implementer.

SNMP operations can be time consuming if a large number of attributes are specified in a query or if the bulk retrieval of tables is attempted.

8.1.6 Question 3: Evaluation of the Management Framework's Impact on the Network

This question deals with the evaluation the management framework's impact on the network.

Evaluation Strategy

The management framework's impact on the network was assessed in terms of the number of packets and the number of bytes generated by the management framework. This includes not only the information generated by the manager and the agent themselves, but also the packet headers and additional acknowledgements generated by the underlying protocols. The measurements have been taken by running the *etherfind* utility

and observing the number of packets and bytes generated in response to particular queries.

The evaluation is facilitated since with reference to packet and byte requirements, the management operations can be broken up into two groups: the one-query-one-object type of query and the one-query-multiple-objects type of query, since it is this aspect of a query that has the principal impact on the number of packets and bytes that will be required to complete the transaction. The conclusions drawn from this study are shown below.

Conclusions

The conclusions are:

1. For one-query-one-object interactions, CMIP requires a greater number of packets and bytes than SNMP due to the additional acknowledgement packets that are generated and the longer packet headers used by TCP.
2. For the one-query-multiple-objects interactions, CMIP takes fewer packets and bytes than when the task is done in SNMP via the GET-NEXT service. This happens because in this case the CMIP agent can put multiple objects in a single packet, thus decreasing the total overhead due to packet headers and acknowledgement packets. However, the number of bytes generated by the manager and agent themselves is lower for SNMP.
3. In general, it is seen that a connection-oriented management protocol produces the least impact on the network and has a minimum overhead due to packet headers and acknowledgement packets when the one-query-multiple-objects approach is used. The ability to use this technique is one of the greatest advantages of the connection-oriented network management protocol. However, the underlying transport connection is also highly prone to failure at the higher rates of packet loss when this approach is used to retrieve relatively large quantities of data.

8.2 Recommendations

The findings of this study lead to a set of recommendations. These have been divided into four categories as shown in the following paragraphs.

8.2.1 Recommendations for Users of Network Management Protocols

The design of a network management protocol allows some operations to be performed more quickly than others. The protocol's behavior in a lossy network is also dependent on its design, especially on whether it is a connection-oriented or a connectionless protocol. These factors have led to the following recommendations.

1. When the network is very lossy (loss rate $> 50\%$), the retrieval of large tables via the linked reply mechanism should be avoided.
2. If two objects are required, and they are in different parts of the MIB tree, separate queries should be used whenever possible. This avoids scoping which may be expensive for the agent to perform.
3. If the two separate objects are not too far apart in the MIB tree, then scoping and linked replies may be used to retrieve them, since linked replies produce the minimum impact on the network.
4. When the net is lossy, the queries to be used should be decided on in advance. This will minimize the number of queries that are used. Since the underlying CO transport connection is lossy, the chances of losing the connection increases with the length of the dialog.
5. The preceding items pertained to connection-oriented protocols. There are no particular recommendations for the users of connectionless protocols like SNMP. However, assuming that such a protocol does not allow scoping, it would be best to manipulate as many objects as possible with a single query, since that would save the overhead associated with multiple datagrams. If scoping is allowed, the cost of the scoping function would influence this decision.

8.2.2 Recommendations for Designers of Network Management Protocols

Proper design of the network management protocol is key to providing satisfactory performance over the entire range of services offered by the protocol. The design philosophy also affects the host impact, network impact, and the protocol's behavior in a lossy network. The results of this study have led to the following recommendations which may be applied during the design of new protocols. They may also be used to improve current protocols like CMIP and SNMP.

1. The Remote Operations Service (ROS) should be used sparingly. This can be achieved by cutting down the number of different operations and distinct PDU types used by the protocol. SNMP is an elegant example of this philosophy, since it specifies only two different PDU structures. It should be noted that a redesigned SNMP would need only a single PDU type to serve every function. Because of the heavy price extracted by ROS, it would be worthwhile to design a network management protocol within the ISO framework that does not use ROS at all.
2. The types of ASN.1 structures allowed by the SMI should be kept at a minimum. Experience with large MIBs would suggest which SMI constructs are necessary.
3. Since the upper ISO layers are resource intensive, the management protocol may be moved down to function over a lower layer.
4. The semantic gap between the representation of data on the physical link and the representation required by the management protocol itself has to be reduced.
5. The management protocol should be able to switch between connection-oriented and connectionless modes of operation depending on the task at hand and the lossiness of the underlying network.
6. a connectionless CMIP may be more advantageous than the connection-oriented CMIP.
7. SNMP may be redesigned to use only one PDU type and to reduce the number of packets needed to transfer large tables.

8.2.3 Recommendations for Implementors of Network Management Protocols

The following recommendations will be useful in implementing network management protocols in an efficient manner.

1. Since all the bottlenecks cannot be determined until a prototype is built, an implementation should be streamlined over several iterations.
2. Kernel data should be made easily accessible to the agent, since a large number of kernel accesses can dramatically increase the times required by functions that require a large amount of kernel data. This will require the kernel's cooperation.

3. The number of data translations should be lowered by using suitable data structures. This is one approach to limiting the semantic gap mentioned previously.

8.2.4 Recommendations for Future Study

This study has identified a number of avenues for further research. These will be outlined here.

1. The total number of packets and the time required to complete an activity in the presence of variable retransmission timeouts and connection failures requires analysis and modeling. Markov processes or timed Petri nets could provide the tools for this purpose.
2. The linked reply mechanism used by connection-oriented management protocols remains to be modeled.
3. It has been observed that $\frac{m}{p}$, where m is the number of packets required to complete a transaction under conditions of no loss and p is the probability of a packet reaching its destination, is a good approximation for the number of packets required in a lossy network to complete an activity over a connection-oriented protocol. This was shown to be true for the data transfer (one-query-one-object) and association establishment phase for high values of p . Further investigation is required before one can precisely state under what conditions this approximation is applicable.
4. This is one of the first attempts to evaluate an upper layer protocol. This field has received little attention until now and is therefore open to further analysis, and the possible introduction of other evaluation methods.

BIBLIOGRAPHY

Bibliography

- [1] ISO 9595: Information processing systems - Open Systems Interconnection - Common Management Information Service Definition, 28 September 1989.
- [2] ISO 9596: Information processing systems - Open Systems Interconnection - Common Management Information Protocol Specification, 28 September 1989.
- [3] Rose, M. T., *The Open Book: A Practical Perspective on OSI*, Prentice-Hall, Inc., 1989.
- [4] Cerf, V., *IAB Recommendations for the Development of Internet Network Management Standards*, RFC 1052, Network Information Center (NIC), SRI International, Menlo Park, CA, April 1988.
- [5] Cronk, R. N., P. H. Callahan, and L. Bernstein, *Rule-Based Expert Systems for Network Management and Operations: An Introduction*, IEEE Network, Vol. 2, No. 5, September 1988.
- [6] Case, J. D., J. R. Davin, M. S. Fedor, and M. L. Schoffstall, *Introduction to the Simple Gateway Monitoring Protocol*, IEEE Network, Vol. 2, No. 2, March 1988.
- [7] Rabie S., A. Rau-Chaplin, and T. Shibahara, *DAD: A Real-Time Expert System for Monitoring of Data Packet Networks*, IEEE Network, Vol. 2, No. 5, September 1988.
- [8] Zhan W., S. Thanawastien, and L. M. L. Delcambre, *SIMNETMAN: An Expert Workstation for Designing Rule-Based Network Management Systems*, IEEE Network, Vol. 2, No. 5, September 1988.
- [9] Feridun M., M. Leib, M. Nodine, and J. Ong, *ANM: Automated Network Management System*, IEEE Network, Vol. 2, No. 2, March 1988. pp. 13-19.

- [10] Hinden R., *Host Monitoring Protocol*, RFC 869, Network Information Center (NIC), SRI International, Menlo Park, CA, December 1983.
- [11] Partridge C. and G. Trewitt, *The High-Level Entity Management System (HEMS)*, RFC 1021, Network Information Center (NIC), SRI International, Menlo Park, CA, October 1987.
- [12] Partridge C. and G. Trewitt, *The High-Level Entity Management Protocol (HEMP)*, RFC 1022, Network Information Center (NIC), SRI International, Menlo Park, CA, October 1987.
- [13] Partridge C. and G. Trewitt, *HEMS Monitoring and Control Language*, RFC 1023, Network Information Center (NIC), SRI International, Menlo Park, CA, October 1987.
- [14] Partridge C. and G. Trewitt, *HEMS Variable Definitions*, RFC 1024, Network Information Center (NIC), SRI International, Menlo Park, CA, October 1987.
- [15] Case, J. D., J. R. Davin, M. S. Fedor, and M. L. Schoffstall, *Network Management and the Design of SNMP*, *ConneXions*, 3(3), March 1989.
- [16] Case, J. D., J. R. Davin, M. S. Fedor, and M. L. Schoffstall, *A Simple Network Management Protocol*, RFC 1157, Network Information Center (NIC), SRI International, Menlo Park, CA, May 1990.
- [17] McCloghrie, K. and M. Rose, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1213, Network Information Center (NIC), SRI International, Menlo Park, CA, March 1991.
- [18] Warrior, U. and L. Besaw, *The Common Management Information Services and Protocol over TCP/IP (CMOT)*, RFC 1095, Network Information Center (NIC), SRI International, Menlo Park, CA, April 1989.
- [19] Warrior, U., L. Besaw, L. LaBarre, and B. D. Handspcker, *The Common Management Information Services and Protocols for the Internet (CMOT and CMIP)*, RFC 1189, December 1989.
- [20] Rose, M., *ISO Presentation Services on top of TCP/IP-based internets*, RFC 1085, Network Information Center (NIC), SRI International, Menlo Park, CA, December 1988.

- [21] ISO/IEC DP 10165-1: Information Processing Systems - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 1: Management Information Model, 27 June, 1989.
- [22] Rose, M., and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based internets*, RFC 1065, Network Information Center (NIC), SRI International, Menlo Park, CA, August 1988.
- [23] Rose, M. T. and D. E. Cass, *ISO Transport Services on top of the TCP*, RFC 1006, Network Information Center (NIC), SRI International, Menlo Park, CA, May 1987.
- [24] MacLeod, J. N., *Performance of Encoding/Decoding the Transfer Syntax*, Tenth Annual International Phoenix Conference on Computers and Communications, 1991.
- [25] Zatti, S. and P. Janson, *Interconnecting OSI and non-OSI Networks using an Integrated Directory Service*, Computer Networks and ISDN Systems, Volume 15 (1988). pp. 269-283.
- [26] Shue, C., W. Haggerty, and K. Dobbins, *OSI Connectionless Transport Services on top of the UDP*, Version: 1, RFC xxxx, submitted to the IETF for review, undated.
- [27] Cerf, V., *Report of the Second Ad Hoc Network Management Review Group*, RFC 1109, Network Information Center (NIC), SRI International, Menlo Park, CA, August 1989.
- [28] Postel, J., *NCP/TCP Transition Plan*, RFC 801, Network Information Center (NIC), SRI International, Menlo Park, CA, November 1981.
- [29] LaBarre, L. (editor), *OSI Internet Management: Management Information Base*, RFC 1214, Network Information Center (NIC), SRI International, Menlo Park, CA, April 1991.
- [30] Rose, M.T. (editor), *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1158, Network Information Center (NIC), SRI International, Menlo Park, CA, May 1990.
- [31] Stine, R. (editor), *FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices*, RFC 1147,

Network Information Center (NIC), SRI International, Menlo Park, CA, April 1990.

- [32] Bitton, D. and C. Turbyfill, *A Retrospective on the Wisconsin Benchmark*, Readings in Database Systems (Stonebraker, M. ed.), Morgan Kaufmann Publishers, Inc., San Mateo, California. pp. 280-299.
- [33] Anon et al, *A Measure of Transaction Processing Power*, Readings in Database Systems (Stonebraker, M. ed.), Morgan Kaufmann Publishers, Inc., San Mateo, California. pp. 300-312.
- [34] Ghatak, S., *An Investigation into the Speed and Host Impact of Various Transmission Protocols*, Technical Report 88-75, The University of Tennessee, Knoxville, August 1988.
- [35] Ferrari, D., G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*, Prentice-Hall, Inc., 1983.
- [36] Dongarra, J., J. Martin, and J. Worlton, *Computer benchmarking: paths and pitfalls*, IEEE Spectrum, July 1987. pp. 38-43.
- [37] Turbyfill, C., *Comparative Benchmarking of Relational Database Systems*, Ph. D. thesis, Technical Report 87-871, Department of Computer Science, Cornell University, Ithaca, New York, September 1987.
- [38] Rose, M. T., *The Simple Book: An Introduction to Management of TCP/IP - based internets*, Prentice-Hall, Inc., 1991.
- [39] Jain, R., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, John Wiley & Sons, Inc., 1991.
- [40] Ben-Artzi, A., A. Chandna, and U. Warrier, *Network Management of TCP/IP Networks: Present and Future*, IEEE Network Magazine, July 1990. pp. 35-43.
- [41] Carter, E. and J. Dia, *Evaluating Network Management Systems: Criteria and Observations*, Proceedings of the IFIP TC6/WG6.6 Second International Symposium on Integrated Network Management, Washington, D.C., April 1991. pp. 213-223.

- [42] Sanghi, S., S. Sengupta, A. Chandna and G. Wetzel, *How Well do SNMP and CMOT Meet IP router Management Needs?*, Proceedings of the IFIP TC6/WG6.6 Second International Symposium on Integrated Network Management, Washington, D.C., April 1991. pp. 285-300.
- [43] Graham, S. L., P. B. Kessler and M. K. McKusick, *gprof: A Call Graph Execution Profiler*, Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, SIGPLAN Notices, Vol. 17, No. 6, June 1982. pp. 120-126.
- [44] McKusick, M. K., *Re: gprof*, electronic mail communication with the author, 31 May, 1991.
- [45] Kemeny, J. G. and Snell, J. L., *Finite Markov Chains*, D. Van Nostrand Company, Inc., 1960.
- [46] Sauer, C. H. and MacNair, E. A., *Simulation of Computer Communication Systems*, Prentice Hall Inc., 1983.
- [47] Fraser, M. D., *Simulation*, IEEE Potentials, February 1992.
- [48] Comer, D., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice Hall Inc., 1988.
- [49] Kleinrock, L., *Queuing Systems, Volume II: Computer Applications*, John Wiley & Sons, 1976.
- [50] Garg, K., *An Approach to Performance Specification of Communication Protocols Using Timed Petrinets*, Proc. 4th International Conf. on Distributed Computer Systems, San Francisco, CA, 1984.
- [51] Zuberek, W. M., *Modified D-Time Petri Nets, Timeouts, and Modelling of Communication Protocols*, Proc. 6th International Conf. on Distributed Computer Systems, Cambridge, MA, 1986.
- [52] Rose, M., K. McCloghrie, and J. Davin, *Bulk Table Retrieval with the SNMP*, RFC 1187, Network Information Center (NIC), SRI International, Menlo Park, CA, October 1990.
- [53] Satz, G., *A New View on Bulk Retrieval with SNMP*, The Simple Times, Dover Beach Consulting, Inc., 420 Whisman Court, Mountain View, CA, March/April, 1992.

- [54] Liu, W., *Generic Models for Performance Evaluation of Computer Networking Protocols*, PhD Thesis, Georgia Institute of Technology, February 1992.
- [55] Key, K. W., *The Implementation of the Simple Gateway Monitoring Protocol*, Masters Thesis, The University of Tennessee, Knoxville, August 1990.
- [56] Baker, J., electronic mail communication with the author, 11 May, 1992.
- [57] Howes, T., electronic mail communication with the author, 7 May, 1992.
- [58] Reiser, M., *Communication-System Models Embedded in the OSI-Reference Model, a Survey*, Proceedings of the IFIP WG 7.3 International Seminar on Computer Networking and Performance Evaluation, North-Holland 1986. pp. 85-111.
- [59] Meister, B.W., *A Performance Study of the ISO Transport Protocol*, IEEE Transactions on Computers, March 1991. pp. 253-262.
- [60] Mehrpour, H. and A. E. Karbowiak, *Modelling and Analysis of DoD TCP/IP Protocol using Numerical Petri Nets*, IEEE Region 10 Conference on Computer and Communication Systems, Conference Proceedings, Vol. 2, 1990. pp. 617-622.
- [61] Benice, R. J. and A. H. Frey Jr., *An Analysis of Retransmission Systems*, IEEE Transactions on Communications, December 1964. pp. 135-145.
- [62] Choi, J. and C. K. Un, *On Acknowledgement Schemes of Sliding Window Flow Control*, IEEE Transactions on Communications, November 1989. pp. 1184-1191.
- [63] Bae, J. J., T. Suda and N. Watanabe, *Evaluation of the Effects of Protocol Processing Overhead in Error Recovery Schemes for a High-Speed Packet Switched Network: Link-by-Link versus Edge-to-Edge Schemes*, IEEE Journal on Selected Areas in Communications, December 1991, pp. 1496-1509.
- [64] Huitema, C. and A. Doghri, *Defining Faster Transfer Syntaxes for the OSI Presentation Protocol*, ACM Computer Communication Review, October 1989. pp. 44-55.
- [65] Conway, A. E., *A Perspective on the Analytical Performance Evaluation of Multilayered Communication Protocol Architectures*, IEEE Journal on Selected Areas in Communications, January 1991. pp. 4-14.
- [66] Yang, O. W. W., X. X. Yao, and K. M. S. Murthy, *Modeling and Performance Analysis of File Transfer in a Satellite Wide Area Network*, IEEE Journal on Selected Areas in Communications, February 1992.

- [67] Nishida, T., *End-to-End Performance Modeling for Distributed Network Management Systems*, IEEE INFOCOM'91, The Conference on Computer Communications, Proceedings, 1991. pp. 121-129.
- [68] Cerf, V., *The Catenet Model for Internetworking*. Internet Engineering Note 48, Network Information Center (NIC), SRI International, Menlo Park, CA, July 1978
- [69] Rudin, H., *Protocol Engineering: A Critical Assessment*, Protocol Specification, Testing, and Verification: Proceedings of the IFIP WG6.1 International Conference on Protocol Specification, Testing, and Verification, 1988. pp. 3-16.
- [70] Black, U., *Network Management Standards: The OSI, SNMP and CMOL Protocols*, McGraw-Hill, 1992.
- [71] Svobodova, L., *Implementing OSI Systems*, IEEE Journal on Selected Areas in Communication, September 1989. pp. 1115-1130.
- [72] Svobodova, L., *Measured Performance of Transport Service in LANs*, Computer Networks and ISDN Systems, 24 November 1989. pp. 31-45.
- [73] Postel, J., *User Datagram Protocol*, RFC 768, Network Information Center (NIC), SRI International, Menlo Park, CA, August 1980.

Vita

Subhendu Ghatak was born in Calcutta, India on June 15, 1961. He attended St. Agnes Convent School and Hijli High School at Kharagpur. In 1980, he entered the Indian Institute of Technology at Kharagpur and in 1985 received the degree of Bachelor of Technology (Honours) in Computer Science. He entered the University of Tennessee at Knoxville in September, 1985 and received the Master of Science degree in Computer Science in 1987. He continued his studies at the University of Tennessee, Knoxville and was awarded the Doctor of Philosophy degree in Computer Science in December, 1992.

He plans to return to India in order to participate in ongoing telecommunications projects in that country.