Management and processing of network performance information

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: [https://dspace.lboro.ac.uk/2134/10361](https://dspace.lboro.ac.uk/2134/10361)

Publisher: © Omar Bashir

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
<table>
<thead>
<tr>
<th>Vol. No.</th>
<th>Class Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

LoAN COPY
Management and Processing of Network Performance Information

by
Omar Bashir

A doctoral thesis submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University

January 1998

© by Omar Bashir 1998
Dedicated to my son

Shmeer Omar
Abstract

Intrusive monitoring systems monitor the performance of data communication networks by transmitting and receiving test packets on the network being monitored. Even relatively small periods of monitoring can generate significantly large amounts of data. Primitive network performance data are details of test packets that are transmitted and received over the network under test. Network performance information is then derived by significantly processing the primitive performance data. This information may need to be correlated with information regarding the configuration and status of various network elements and the test stations.

This thesis suggests that efficient processing of the collected data may be achieved by reusing and recycling the derived information in the data warehouses and information systems. This can be accomplished by pre-processing the primitive performance data to generate Intermediate Information. In addition to being able to efficiently fulfil multiple information requirements, different Intermediate Information elements at finer levels of granularity may be recycled to generate Intermediate Information elements at coarser levels of granularity. The application of these concepts in processing packet delay information from the primitive performance data has been studied.

Different Intermediate Information structures possess different characteristics. Information systems can exploit these characteristics to efficiently re-cycle elements of these structures to derive the required information elements. Information systems can also dynamically select appropriate Intermediate Information structures on the basis of queries posted to the information system as well as the number of suitable Intermediate Information elements available to efficiently answer these queries.

Packet loss and duplication summaries derived for different analysis windows also provide information regarding the network performance characteristics. Due to their additive nature, suitable finer granularity packet loss and duplication summaries can be added to provide coarser granularity packet loss and duplication summaries.
Acknowledgements

I am indebted to my supervisor, Dr. David Parish, for his guidance, support and encouragement throughout my stay at Loughborough. It would have been impossible for me to complete this research without his help and motivation.

I would like to mention that extensive network performance data were required to test the accuracy and the performance of tools developed here with respect to the existing tools and techniques. I am grateful to British Telecom plc. for allowing me to operate their network performance monitoring systems and to use their existing data processing tools to achieve this objective.

I am grateful to Dr. Iain Phillips for guiding me through the practical requirements of this research project and providing valuable suggestions and pointers throughout this period.

I am extremely thankful to all the members of the High Speed Networks group at Loughborough for helpful and extremely informative discussions, comments and feedback during various phases of this project.

Finally I wish to express my gratitude to my wife Amna, our son Shmeer and our parents for their patience and encouragement.
<table>
<thead>
<tr>
<th>Abbreviations &amp; Acronyms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>CDV</td>
<td>Cell Delay Variation</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>CTD</td>
<td>Cell Transfer Delay</td>
</tr>
<tr>
<td>DBMS</td>
<td>Data Base Management System</td>
</tr>
<tr>
<td>DSS</td>
<td>Decision Support System</td>
</tr>
<tr>
<td>EIS</td>
<td>Executive Information Systems</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JDK</td>
<td>Java Developer's Kit</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>OLAP</td>
<td>On Line Analytical Processing</td>
</tr>
<tr>
<td>OLTP</td>
<td>On Line Transaction Processing</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>RRL</td>
<td>Remote Reference Layer</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SMDS</td>
<td>Switched MultMegabit Data Service</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>
## Contents

Abstract

Acknowledgements

Abbreviations & Acronyms

Contents

### Chapter 1

**Introduction**

1.1 Monitoring Data Communication Networks

1.2 Monitors

1.2.1 Classification of Monitors on the Basis of Implementation

1.2.1.1 Hardware Monitors

1.2.1.2 Software Monitors

1.2.1.3 Firmware Monitors

1.2.1.4 Hybrid Monitors

1.2.2 Classification of Monitors on the Basis of Data Collection Techniques

1.2.2.1 Implicit Data Collection

1.2.2.2 Explicit Data Collection

1.2.2.3 Probing

1.3 Architectures for Distributed Systems Monitors

1.3.1 Layered Architecture

1.3.2 A Distributed Object Oriented Model Based on Monitoring Activities

1.3.3 An Object Oriented Architecture for Adaptive Monitoring of Networks

1.4 Measures of Network Performance

1.5 Intrusive Network Performance Monitoring

1.6 Characteristics of Primitive Network Performance Data

1.6.1 Data Volume

1.6.2 Database Operations

1.6.3 Chronological Ordering of Data

1.6.4 Missing and Suspect Data

1.7 Information Processing and Analysis in Monitoring Systems
1.8 Objectives

Chapter 2
Data Warehousing and Application to Network Monitoring Data

2.1 Introduction to Data Warehouses
2.2 An Architecture for a Warehouse System
  2.2.1 An Architecture for a Data Warehouse
  2.2.2 An Architecture for a Data Warehousing System
2.3 Multidimensional Data Model for Data Warehouse
2.4 Issues in Data Warehouse Design and Development
  2.4.1 Granularity
  2.4.2 Purging Warehouse Data
  2.4.3 Change Detection, Translation and Integration
  2.4.4 Cyclicity of Data
  2.4.5 Performance Optimisations
  2.4.6 Provisioning of Auxiliary Data
    2.4.6.1 Simple Contextual Information
    2.4.6.2 Complex Contextual Information
    2.4.6.3 External Contextual Information
2.5 Distributed Data Warehouses
  2.5.1 Requirements for Local Processing
  2.5.2 Economics of Implementation
  2.5.3 Warehousing Data of Disjoint Activities
  2.5.4 Data Marts
    2.5.4.1 Data Loading
    2.5.4.2 Data Model
    2.5.4.3 Capacity Management
    2.5.4.4 External Data
    2.5.4.5 Performance
    2.5.4.6 Security
  2.5.5 Data Warehouse on Internet/Intranet
2.6 Warehousing Scientific Data
2.7 Warehousing Network Monitoring Data
  2.7.1 Management of Detailed Network Performance Data
  2.7.2 Management of Monitoring Experiment Configuration Data
  2.7.3 Management of Summaries and Pre-processed Information
2.7.4 Management of Auxiliary Data 49
2.8 Users of Information Systems and Data Warehouses 50
2.9 Summary 51

Chapter 3
The Application of Intermediate Information Concepts to Process Packet Delay Measurements 55
3.1 Introduction 55
3.2 Basic Concepts of Intermediate Information 57
3.3 A Model for Information Systems Based on Intermediate Information 59
3.4 Employing Intermediate Information Structures for Packet Delay Information 63
3.5 Intermediate Information Structure for Packet Delay Information 66
   3.5.1 Sorted Delays as Intermediate Information 67
   3.5.2 Delay Distributions as Intermediate Information 67
3.6 Comparison of Information Systems Using Sorted Delays and Delay Distributions 68
3.7 An Architecture for Network Performance Information Sub-system Employing Intermediate Information 73
3.8 The Impact of Information Reusability on the Performance of Information System 76
3.9 Related Research 79
3.10 Conclusions 81

Chapter 4
Dynamic Selection of Intermediate Information Structures 83
4.1 Introduction 83
4.2 Related Research 85
   4.2.1 Self-Organising Data Structures 85
   4.2.2 Database Re-Organisation 87
   4.2.3 Query Optimisation 90
4.3 Approaches to Dynamic Selection of Intermediate Information Structures 91
   4.3.1 Independent Management of Sorted Delays and Delay Distributions 92
   4.3.2 Deriving Delay Distributions from Sorted Delays 94
4.4 A Prototype System Employing Dynamic Selection of Intermediate Information Structures

4.4.1 Query Cost Calculation 99
4.4.2 System Architecture 103
  4.4.2.1 QueryPlanOptimiser Class 105
  4.4.2.2 DelayQueryProcessor Class 108

4.5 Re-using and Re-cycling Coarse Granularity Delay Distributions 110

4.6 Conclusions 114

Chapter 5
Management of Packet Loss and Duplication Summaries 118

5.1 Introduction 118
5.2 Characteristics of Packet Loss and Duplication Data 119
5.3 Modelling Packet Counts Data 121
5.4 Calculation of Packet Counts Summaries from the Primitive Database 125
5.5 Management of Packet Counts Summaries 128
5.6 Summary and Discussions 131

Chapter 6
A Server for Network Performance Information 134

6.1 Introduction 134
6.2 Query and Information Objects 134
  6.2.1 Query Objects 135
    6.2.1.1 QuerySpecification Class 135
    6.2.1.2 AggregateDelayQuerySpecs Class 136
    6.2.1.3 FinalResultsQuerySpecs Class 136
  6.2.2 Information Objects 136
    6.2.2.1 PrimitiveData2 Class 137
    6.2.2.2 DerivedResults Class 139
    6.2.2.3 FinalResults Class 140
6.3 Server Architecture 141
  6.3.1 Remote Server Object 142
    6.3.1.1 PrimitiveDataCollector2 Class 142
    6.3.1.2 DerivedResultsCollector Class 144
  6.3.2 Server Manager 146
6.4 Example Client 147
6.5 Summary 148
## Chapter 7

**Conclusions and Directions for Future Research**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Discussions and Conclusions</td>
<td>151</td>
</tr>
<tr>
<td>7.2 Directions for Future Research</td>
<td>157</td>
</tr>
<tr>
<td>7.2.1 Cache Management</td>
<td>157</td>
</tr>
<tr>
<td>7.2.2 Pre-emptive Pre-processing</td>
<td>159</td>
</tr>
<tr>
<td>7.2.3 Client Data Caching and Exploitation of Client Resources</td>
<td>160</td>
</tr>
</tbody>
</table>

## References

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>162</td>
</tr>
</tbody>
</table>

## Bibliography

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>176</td>
</tr>
</tbody>
</table>

## Appendix A

**Object Oriented Software Design**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
</tr>
</tbody>
</table>

## Appendix B

**Client Server Computing**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>194</td>
</tr>
</tbody>
</table>

## Appendix C

**Client Server Databases**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>206</td>
</tr>
</tbody>
</table>

## Appendix D

**Object Wrapper for Primitive Network Performance Data**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>215</td>
</tr>
</tbody>
</table>

## Appendix E

**High Level Network Performance Information: Incidents and Effects**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>225</td>
</tr>
</tbody>
</table>

## Appendix F

**Framework for Simulating Information Systems Based on Intermediate Information**

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>238</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Monitoring Data Communication Network

Computer communication networks and distributed systems present serious management challenges to the users and the service providers. Network service providers generally strive to provide an acceptable performance to the users by attempting to manage the network resources in an optimum manner.

With the development of massive internetworks that employ heterogeneous technologies, the art of network management is also being formalised and standardised. Functional areas critical to network management have been identified and addressed at a number of occasions. These functional areas generally relate to the following [BruSto89], [Slo95]

- management of network configuration
- accounting for service utilisation and customer billing
- security against unauthorised access
- detection of system faults
- management and optimisation of network performance

Research has also been conducted towards an integrated network management approach. This has primarily been motivated by the desire to extend management activities beyond the vendor and technological boundaries [Rab92]. Such integrated systems also hold the promise to manage scalability in the managed network as well as the scope of management activity. Moreover the application of expert systems and knowledge based systems to network monitoring and management has also been studied [RabRS88].

Network management systems monitor the status of the network components. All network management functions depend upon the data generated by monitoring the services for effective management [SloMof89]. Thus network monitoring systems form the basic elements of a management system. These systems attempt to determine the utilisation of the network being monitored and the performance of its various components. These systems can also detect out of control situations and generate suitable alarms on the occurrence of these instances. The data acquired by these systems and the information analysed may be retrieved by the service
providers at regular intervals to obtain summaries on network performance and their utilisation [EthSim92], [Hel92].

Packet delay characteristics of data communication networks are determined from measurements collected by dedicated monitoring equipment which generate test traffic on the network links under test. Current management information bases (MIBs) do not manage and process data related to the packet delay characteristics of the networks [Lam95].

The monitored data and subsequently derived information may also be used for some general network management activities [SamSlo93]. This means that the performance information in association with the information regarding the characteristics and the configuration of the monitored network and its components can be used to make management decisions and to perform appropriate control actions (Figure 1-1)

![Figure 1-1: Network Monitoring & Associated Activities (Adapted from [SamSlo93])](image)

It has been suggested that it is the quality of information, rather than mere monitored data, and the functionality, reliability and usability of the monitoring tools that guarantee effective management [Rab92]. On the other hand inappropriate or inaccurate analysis may provide the network managers as well as the users with inconsistent and unreliable indication of network performance. Thus the success of the entire operation depends upon efficiently maintaining and
appropriately processing the primitive network monitoring data for subsequent analysis in relation to the network configuration information.

Even at modest sampling rates, the data collected by the monitoring devices grow at a phenomenal rate. Network performance analysis is usually an iterative process. The initial analysis of the summaries derived from the primitive data generates further information requirements. For example the analyst may want to *zoom into* a detected event by deriving the required information at finer levels of granularity. This would allow a detailed examination of the event being studied. Alternately it may be desirable to *zoom out* by deriving the required summaries at a coarser level of granularity for a longer period. This allows correlating the detected events with the previously monitored network effects. In other cases, the derivation of different summary elements may be desirable. For example, if the average delay value exceeds a preset threshold, the analyst may need to view the average delay value for the slowest 10% packets and the range of packet delay values for that analysis window.

It is not possible to pre-process all the different types of summaries at various levels of granularity. Deriving these summaries from primitive data each time they are required can be an extremely expensive process which can have a negative impact on the performance of the complete analysis process.

This thesis describes techniques that can be used to pre-process network performance data collected by intrusively monitoring data communication networks. The information resulting from the pre-processing operations is known as *Intermediate Information*. These Intermediate Information elements derived at the finest granularity (for example one hour) can be *re-used* to generate Intermediate Information elements at coarser levels of granularity (for example 3 hours or 6 hours). Moreover various information elements (for e.g., averages, variances, percentiles etc.) can be derived efficiently from these Intermediate Information elements. As accesses to primitive data are expected to reduce, the performance of the database and the efficiency of the analysis operations is expected to increase significantly.

This chapter briefly describes the principles of network monitoring and the characteristics of intrusively monitored network performance data and the associated processing applications. Some of the problems in maintaining and
processing this data to provide the required information are then discussed. In the end, the objectives of this work and the organisation of this thesis are explained.

1.2 Monitors
Jain defines monitors as tools or sets of tools to observe the activities on a system [Jai91]. The task of monitoring systems is generally not limited to simple observation. These systems perform varying degrees of processing and analysis on the observed data and are required to provide summaries to the system user or the system manager either on request or on regular basis. It is also not unusual to find monitoring systems as a component in the control loop of a system. When such monitoring systems detect a situation that might lead to an out of control scenario, they may trigger a compensation module to provide necessary stimulus to the system so that such a scenario may be avoided.

1.2.1 Classification of Monitors on the Basis of Implementation
Monitors are generally divided on the basis of implementation strategies. Monitors may be classified into the following four basic types depending upon their implementation.

1.2.1.1 Hardware Monitors
These monitors generally have probes that may be connected to the monitored system at convenient locations. These monitors can observe the system operation without affecting the dynamic behaviour of the system. In the process they neither cause any interference in the monitored system operation nor do they require any assistance from monitored systems in terms of resources on the monitored system. The major drawback with such monitors is the inability or the difficulty of associating the monitored behaviour to the software of the monitored system.

1.2.1.2 Software Monitors
Software monitors are special measuring and monitoring programs that are incorporated into the software of monitored system. These monitors seriously affect the dynamic behaviour of the monitored system and depend on the resources of the monitored system as well. Results generated by these monitors can however be conveniently linked to the software as well as the data being used by the monitored system.
1.2.1.3 Firmware Monitors
Jain defines firmware monitors as ones that are implemented in the processor microcode [Jai91]. Firmware monitors are generally suited for systems where software monitors cannot be used because of strict timing considerations and hardware monitors cannot be connected due to inaccessibility or non-availability of probe/test points. Sometimes firmware monitors are considered similar to software monitors.

1.2.1.4 Hybrid Monitors
Hybrid monitors consist of a combination of hardware, software and firmware monitoring facilities. The software modules in these systems reside on the monitored system to generate the measured data that is made available to external hardware. This external hardware also physically probes the monitored system to acquire the complement of the data furnished to it by its software counterpart. Such monitors possess the excellent data reduction facilities of the software monitors and high resolution that is a characteristic of the hardware monitors [Jai91].

1.2.2 Classification of Monitors on the Basis of Data Collection Techniques
Monitors and monitoring systems may also be classified on the basis of the data collection techniques that they employ. Monitors may make observations on the occurrence of a certain event (event driven measurements). Measurements may also be carried out at a regular or random rate by sampling the monitored resource on the monitored device or system (Time driven measurements or sampling). Sampling is usually carried out without any synchronisation with the activities of the monitored device.

Depending upon the resources on the monitored device being measured as well as the data that is being collected, a monitoring system may employ one or a combination of the following data collection techniques.

1.2.2.1 Implicit Data Collection
Also known as implicit spying [Jai91], this is the least intrusive technique and requires promiscuous observation of the activity on the monitored system. This technique presents minimum impact on the performance of the system being monitored. Usually implicit spies operate with a filter at the front end so as only to observe the activity or events of interest.
This technique is commonly used to monitor the traffic over networks (for example studies conducted by [Cac89] and [Mah94]). Data generated by implicitly spying over networks may be used to determine the type of traffic over the network, network services being utilised and the type of distributed applications operating at a specific time.

1.2.2.2 Explicit Data Collection
Explicit data collection is generally used to augment the data obtained from implicit spying [Jai91]. This technique requires provisions of trace points or probes in the system and generates some overheads on the system being monitored. Moreover each component of the system that needs to be monitored may have to be instrumented differently.

1.2.2.3 Probing
Probing involves making feeler requests [Jai91] on the system being monitored by generating sample workload over it. Probing has been applied extensively to monitor performance of communication networks (for example, systems developed by [Sid89], [SidPA89], [CarCro96], [PhiPR95] and [PhiTP96]). One of the most common method is to send specially marked packets over the network to the destination. The time required by the packets to transverse the network and the number of packets that are dropped in the process may provide information regarding the performance of different network elements with respect to the traffic on the network. Similarly a combination of packets may be sent to the destination at a specific rate. The variation in the arrival rate may allow inferring the queue lengths over the specific links on the network.

1.3 Architectures for Distributed System Monitors
Jain suggests that monitoring distributed systems is generally more difficult than monitoring a centralised system. Monitors for distributed systems may also be distributed and perform a number of concurrent activities [Jai91].

All monitoring systems must generate as little interference of the monitored system as possible. Distributed monitoring systems may also require a highly accurate and synchronised time base to synchronise the activities of various elements of the monitoring system. A number of approaches in this direction have been explored. Some involve the approximation of the current time at various
nodes by exchanging time values over a large time period. Endriss et al have implemented a global time base by a dedicated link between monitoring stations in NETMON II [EndSZ86]. Many monitoring systems synchronise their activities by acquiring timing information from external global timing agencies, for example the Rugby Clock [Sid89], [SidPA89] and the Global Positioning System [PhiPR95], [PhiTP96].

The task of monitoring a distributed system does not end at data collection. Data collection, as explained later, is at the lowest level of the entire monitoring process. The monitored data has to be processed in order to derive intelligible information. This information is then presented to the service providers in an appropriate manner. Thus development of a comprehensive monitoring system is an enormous task. Such systems may be developed efficiently by modelling their behaviour at a higher level of abstraction.

The following paragraphs explain three architectures for distributed system monitors at the highest level of abstraction. The first architecture presents a layered view of the monitoring system, where each layer provides a service for the layer above it. The second architecture defines the monitoring process as a set of monitoring activities performed in a loosely coupled object based distributed system. The third architecture presents an object based approach to monitoring distributed systems, with an aim to achieve maximum flexibility in system operation as well as subsequent expansion and modification.

1.3.1 Layered Architecture

Jain describes a layered architecture for monitoring systems [Jai91]. In this approach, the entire monitoring system is viewed as a number of layers performing different functions (Figure 1-2). Each layer provides a service to the layer above it. The architecture described here also allows for system management and control.

The observation layer is tasked to collect raw data. It may accomplish this task either by spying implicitly, by explicit instrumentation, by probing or by a suitable combination of any of these. The choice depends upon the parameters being measured as well as the monitored systems.
Collectors gather data observed by the observers. Each collector may gather observed data from a number of observers. Large systems employing a number of observers at distant locations may also have more than one collector.

The analysis layer is tasked with performing analysis operations on the data collected by the collectors, and presenting reasonably higher level information to the layers above it. This layer performs appropriate statistical operations on the collected data to generate the required summaries from the collected data.

The presentation layer provides the user interface to the monitor. This layer generates various reports, displays results of analysis and generates alarms at the occurrence of various out of control events.

The interpretation layer requires the use of the human observer or an expert system to perform meaningful interpretation of the analysed information. This layer may involve activities such as trend analysis, correlation of information derived by measuring different parameters or different elements of the monitored system etc.

---

The console provides an interface to control the system parameters and states. The console may not be considered as a part of the monitoring system. As monitoring and control functions are often used together, it may therefore be desirable to allow the use of system control facilities along with the system observation tools.
Decisions on varying system parameters or configuration are usually carried out by the **management layer**. These decisions are based on the analysis and interpretation of the observed data. The **manager** (the entity performing the management operation) implements its decisions using a console.

A monitoring system may contain multiple components from each of the layers. Jain suggests many-to-many relationships between successive layers, for example a single observer sending data to multiple collectors or a collector collecting data from multiple observers [Jai91] (Figure 1-3).

Jain suggests that observers may employ software, hardware, firmware or hybrid components to make the required observations on various elements of the monitored system [Jai91]. Collectors, analysers and presenters are usually implemented in software. The console layer can either be a software entity implemented over a particular workstation, or may consist of specialised hardware, for example switch boards or control panels etc. Interpreters and managers are usually humans but Jain has not ruled out the possibility of automation of these layers [Jai91].

![Layered Distributed System Monitor: Many to Many Relationship](image)

**Figure 1-3**: Layered Distributed System Monitor: Many to Many Relationship
1.3.2 A Distributed Object Oriented Model Based on Monitoring Activities

This model has been described by Samani and Sloman [SamSl093]. It is based around the following four monitoring activities performed in a loosely coupled object based distributed system,

- The *generation* activity detects important events and generates event and status reports. These monitoring reports are used to construct monitoring traces, which represent the historical views of system activity.
- The *processing* activity performs common processing functions such as merging of traces, validation, database updating, combination / correlation and filtering of monitoring information. It converts the raw and low-level monitoring data to the required format and level of detail.
- The *dissemination* activity governs the distribution of monitoring reports to the users, managers or processing agents who might require them.
- The *presentation* activity allows for the display of the gathered and the processed information in an appropriate manner.

This model may seem to be similar to a layered model. It may be argued that generation is the lowest layer of the model and presentation is the highest layer which uses the services of the lower layers. Samani and Sloman however suggest that generalised monitoring systems may need to perform these activities in various places and in different orders to meet specific monitoring requirements. For example, generated information may be directly displayed by an object without processing or dissemination. Events and reports which are distributed to particular managers could be re-processed to generate new monitoring information or events. Presentation of information may occur at many intermediate stages. It is for this reason the monitoring model has been presented as a set of activities that may be combined as required in a generic monitoring service [SamSl093].

This architecture therefore does not identify the overall controlling activity for the monitoring system. The segregation of different activities is logical but all these activities should be controlled appropriately so that they contribute towards a common goal. It is necessary that an agency should control and coordinate the overall operation of entire system. This agency should also be able to command / suggest variations in the operation of each activity depend upon the user requirements or evolution in the operating environment.
1.3.3 An Object Oriented Architecture for Adaptive Monitoring of Networks

The architecture suggested here is based on the one proposed by [PhiBP97]. This architecture allows monitoring of data communication networks in a distributed and an object oriented manner. This architecture is flexible as additional areas of a network can be monitored by the simple addition of extra *Monitors*. Moreover the control of the entire system is centralised, allowing considerable automation of the system due to the system wide knowledge maintained by a *Director* within a Control Station. Basic objectives are therefore

- scalability
- reusability
- ability to perform adaptive operations

This system consists of a number of *Monitors*, that measure the monitored entities at different geographical locations (Figure 1-4). These *Monitors* may employ any one of the above mentioned data collection techniques. These are controlled by a central controller or a *Director*. The *Director*, in addition to providing a suitable user interface, will manage the complete monitoring system. The *Director* controls the operation of these *Monitors* by

- monitoring the status of every *Monitor*
- initialising suitable *Monitors* to perform tests depending upon the user requirements, the number of *Monitors* available and the status of the monitored system
- upon failure of a particular *Monitor*, perform essential data recovery operation to retrieve the data monitored before the malfunction
- attempt to re-start *Monitors* after a failure

![Diagram of Objects in a Distributed Adaptive Networks Monitoring System](image-url)

Figure 1-4: Objects in a Distributed Adaptive Networks Monitoring System
The **Director** also controls a third entity, the **Store**, which may be implemented as a separate computing device, or as a part of the **Director**. The **Store** is tasked with:

- the storage of monitored data
- the derivation of summaries from the monitored data
- the reception of queries for information from the **Director** and provision of the required information to the **Director**

The system may also employ a set of **Agents** to communicate with other computer-based entities, for example, porting performance information to network management systems.

The control of the system is based on the concept of **Jobs**. These are atomic pieces of work specification, for example, start of test, retrieval of results from monitor process, process queries in the store etc. These **Jobs** may be scheduled as **absolute jobs**, i.e. conducted periodically, or as **relative jobs**, i.e. in relation to other jobs.

These objects may communicate with each other over the monitored network, as shown in figure 1-4. Alternatively they can communicate over a separate network (Figure 1-5). In this case the **Monitors** may retain their connections with the monitored network, whereas the **Director** and the **Store** may use the separate network to communicate amongst themselves and the **Monitors**.

![Object Interaction Over a Separate Management Link](image)

A further aspect of this system is the ability of the **Director** to operate in an automated manner, i.e. without direct requests from the users or the **Agents**.
Examples of this automated behaviour would include the automatic initiation of a special extra or focused test as a result of an abnormal condition detected after processing the data collected from a previously initiated test. The results of the second automated test would then provide more information regarding the suspect area of the network. Moreover, if the Store does not contain sufficient data or information to answer a specific query made by a user or an Agent, a test could be initiated automatically to collect the required information.

1.4 Measures of Network Performance
Average delays and throughput are considered to be the traditional performance measures for packet switched networks. In packet switched networks, delays experienced by data packets are a combination of link delays and switching delays. Link delays are dependent on the length of the packets, speed of the links, network queues, loading and priority associated with the packets. Switching delays, on the other hand, are dependent on the status of the processor at the node, its buffer size, interrupt priority structure as well as the processing power.

For networks supporting real time applications (for e.g. audio and video) some measure of delay variation needs to be calculated in addition to mean delay [Tan96]. For example, in ATM networks cell delay variation (CDV) is used as a standard performance parameter in addition to the cell transfer delay (CTD). Roppel explains 2-point CDV and 1-point CDV [Rop95]. 2-point CDV describes the variability of cell transfer delay introduced by the connection portion between two measurement points. 1-point CDV describes the variability of cell arrival times at a measurement point with respect to the negotiated peak cell rate [Rop95].

Due to the limited link capacity and the finite processing power of the node processors in the packet switched networks, long queues of data packets can build up waiting to be transmitted in case of heavy loading. This can result in significant degradation of network performance [Tan96]. Limited buffer capacity at the nodes results in congestion. Packets incident on the nodes that have no buffer vacancy are discarded, thereby resulting in packet loss.

Congestion may also result due to structural resource imbalance [Tan96]. For e.g., if a high speed communication line is connected to a low speed PC, the CPU may not be able process the incoming packets fast enough. Thus some packets will
be lost. These packets will eventually be re-transmitted, adding to delays, wasting bandwidth and generally reducing performance.

*Synchronously triggered events* can also cause congestion. Tannenbaum explains one such event occurring due to a *broadcast storm* [Tan96].

Packet losses occur due to network faults as well. Additionally errors induced during transmission may require the receiver to discard erroneous packets and request for re-transmission. However errors induced during communication are few and their occurrence is also rare.

Connectionless packet switched networks may also generate duplicate packets which are discarded by the protocol at the destination node or by the application at the host. Duplication of packets may either be a characteristic of the routing algorithm or may occur due to delayed acknowledgement of packet reception at the destination host. In the latter case the source host may time out and, assuming that the packet was lost, will re-transmit the packet again.

Thus packet losses, duplications and packet loss rates may also provide valuable information regarding the characteristics of various network elements. These also allow detection of abnormal conditions such as loss of service or a node failure.

### 1.5 Intrusive Network Performance Monitoring

Intrusive network monitoring systems use special *time stamped test packets* (Figure 1-6) which are transmitted over the network being monitored towards a remote receiver test station. The receiver test station receives the test packets and logs the arrival of the test packet along with the time of reception of the test packet (Figure 1-7a). In some cases the receiver test station may insert the reception time in the received test packet and re-transmit it to the transmitter test station (Figure 1-7b). It is however necessary that clocks at both sites i.e. the transmitter and the receiver test stations, are synchronised.

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Packet Length</th>
<th>Packet ID</th>
<th>Transmit Time</th>
</tr>
</thead>
</table>

Test ID

Figure 1-6: Test Packet Contents
The network test stations may conduct similar such tests for a number of routes over a large network. In the simplest form, following details of each packet transmitted by a transmitter test station may be stored in a table.

- **Test ID.** This is a field or a combination of fields representing the route i.e. source and destination node addresses or IDs, packet length and other test station related parameters. Test ID is inserted in the test packet by the transmitter test station. Test ID may also represent *configuration data* [Sho91] for a particular monitoring experiment.

- **Packet ID.** This is represented by a number used to uniquely identify a test packet. It may be assigned by a simple counter which is incremented each time a test packet is transmitted. Packet ID is also inserted in the test packet by the transmitter test station.

- **Transmit Time and Date.** Time and date when the test packet is transmitted by the transmitter test station. This data may also be inserted in the test packet for transmission to the remote receiving test station.

![Diagram of Intrusive Monitoring](image)

Figure 1-7: Intrusive Monitoring
The receiver test station receives the test packet destined to it and, in addition to the above mentioned data, stores the time when it received the test packet. The difference between the receive time and the transmit time indicates the delay experienced by the test packet as it travelled from the transmitter test station to the receiver test station.

Network performance monitoring is usually conducted at a specific regular rate continuously. The main aim of the entire exercise is to view the behaviour of the network relative to data traffic on it. Such behavioural information may accurately be interpreted after making a sufficient number of observations. Under certain exceptional circumstances, for e.g. error conditions and faults, the tests being carried out by the monitoring / test stations are varied so as to view the affect on other resources (routers, links etc.) of the networks. This may also involve varying the frequency of tests.

Real time determination of a fault or an error condition is generally not conducted by the test stations, as they are usually low performance machines tasked only with the generation of test packets, their reception and maintenance of logs of received and transmitted test packets. Real time fault detection is generally conducted by an another computing entity known as the *alarm station* (Figure 1-8). The receiver test station, upon reception of the test packet, may transmit details of the received test packet to the alarm station. The alarm station may log only sufficient data to perform real time detection of error conditions or faults such as excessive delays, losses etc.

---

**Figure 1-8 : Alarm Station**

- Packet ID
- Test ID
- Transmit Time
- Receive Time
- Delay
The test data logged at the transmitter and receiver test stations are periodically retrieved for off-line analysis over a separate computing platform. These data need to be stored in suitable databases and analysed appropriately to determine various aspects of network behaviour.

Several variations of such basic network performance monitoring operation are possible, for example multiple test packets or burst of test packets may be transmitted over the network being monitored. Such tests may be devised to determine characteristics other than simple delay and loss characteristics of the monitored network.

1.6 Characteristics of Primitive Network Performance Data
Network performance monitoring can be considered as a set of continuous experiments. Each experiment may involve monitoring a particular resource, for example a particular route or device. Data sets related to each experiment are retrieved regularly from the monitors for processing and analysis. Each data set consists of two tables, a table maintained by the transmitter test station and a table maintained by receiver test station. It may also be useful to generate logs of the test station status at regular intervals. These logs of instrumentation data [Sho91] may also be retrieved when the performance data is retrieved from the test station. These data may be used in subsequent analysis to indicate the accuracy and validity of the monitored data and the derived results. Upon retrieval, the data sets are merged with the database of the performance data (Figure 1-9).

![Diagram: Data Set Retrieval & Database Merge](image-url)

**Figure 1-9 : Data Set Retrieval & Database Merge**
In the most basic form the database may represent two tables i.e. one for data regarding the transmitted test packets and the other for data regarding the received test packets. It may be appropriate to partition the database files on the basis of certain criteria, for example test IDs, monitoring periods etc. The data sets retrieved from the monitor stations are actually merged with the appropriate partitions in the database (Figure 1-10).

![Figure 1-10 : Data Set Merge to Database Partitions](image)

For all practical purposes the retrieved data in these databases are considered as primitive monitoring data. It is assumed that the test stations and the control station perform appropriate checks on the integrity of the monitored data. Only the data items that pass these integrity checks are included in the respective tables. At this particular level, the acquired data exhibits following characteristics.

1.6.1 Data Volume
With tests being conducted regularly on a number of routes or network resources, the data collected by the monitoring test stations grow at a colossal rate. Moreover it is generally convenient to retain primitive data because the processing requirements for performance monitoring may be extremely diverse in a relatively dynamic environment. Summarising this data while keeping in view only a few applications may later prove to be a short sighted decision. Primitive monitoring data may therefore need to be retained for significantly long periods of time.
1.6.2 Database Operations

Only two types of operations may occur on the database containing the primitive performance data, i.e. **merges** and **retrievals**. **Deletions** of parts of the database may be carried out once such data are not required or have been archived.

**Merges** are conducted once logs are downloaded from the test stations and added to the database. Records are **retrieved** from the database by the information processing applications. Information processing applications generally request a large number of records from the primitive database to derive the required summaries.

1.6.3 Chronological Ordering of Data

Logs maintained by the transmitting test stations contain packet details recorded automatically in a chronological order. Chronological ordering of the logs retrieved from the receiving test station on the basis of the transmit time may be required for most of the applications. The performance of networks is generally analysed for different analysis windows within the monitoring period (for example 3 hour windows during a day). Thus chronological ordering of data greatly facilitates searching and retrieving the required data from the primitive database.

1.6.4 Missing and Suspect Data

Management of missing or suspect data in the network performance databases may have to be associated with the instrument logs from the respective test stations. Under normal circumstances, packets transmitted by the transmitting test station and not received by the receiving test station are lost due to some fault or performance drawbacks in the network. These lost packets are not considered as missing data.

However, a test station malfunction is characterised by discontinuities in the monitored traces. These are identified from packet losses after examining the instrument logs.

1.7 Information Processing and Analysis in Monitoring Systems

Different network management and monitoring applications impose different information and analysis requirements. For example, activities aimed at detecting
faults in network elements and generating appropriate alarms may require calculation of simple statistical measures on a small subset of monitored data to indicate the occurrence of these abnormal events in real time.

In contrast, off-line analysis of performance data to detect various trends and to generate network performance models may require extensive processing of large amounts of historical monitoring data. Moreover at higher levels of processing, this information may need to be correlated with information regarding the configuration of networks devices being monitored as well as the information regarding the status and operating modes of the monitoring stations. This correlation aids in accurately interpreting the performance information and troubleshooting various faults.

The requirements of different groups of users of the network (users, managers or service providers) for the performance information may also vary significantly [Sid89]. Some information that may be of interest to a certain group of individuals may not be useful to others. In some cases (for e.g. research and development) the processing requirements may be extremely dynamic and may not be known accurately at the system analysis stage of the development. Moreover some events that require detailed investigation may be determined a long time after the monitoring system has been in operation. Other information processing requirements may be generated once the current information processing activities fail to reveal the existing events or do not present necessary information to the users.

Thus in the simplest case, a number of processing applications access a common primitive database to generate the required results (Figure 1-11).

![Diagram of Monitoring System and Applications Using Primitive Database](image)

**Figure 1-11 : Applications Using Primitive Database**
Most of the applications requesting/processing network performance data may proceed through a number of similar processing stages. Additionally a few applications may require information that has been generated by other applications. In such situations, it may be tempting to access the information that has already been processed by other applications to generate the required information elements. For example, average packet delays may be required for a specific performance analysis requirement. Another performance analysis application may need delay variance. It may be efficient to use the averages calculated by the first application along with the primitive data to derive the required delay variance. This scenario is presented in figure 1-12.

![Diagram of information sharing across applications](Image)

**Figure 1-12 : Information Sharing Across Applications**

However it is important to note that most of the information processing applications generally have a significantly shorter life than the data that they process. Some of these applications are developed to satisfy only one time requirement. Other may only be used rarely at specific instances. Some applications may also be modified to provide information elements different from the ones provided by the earlier versions.

Thus designing information processing and analysis applications on the basis of the information derived by other applications may not be appropriate. In addition to resulting in an unstructured information processing environment, it may present a serious risk as incorrect information may be used by these applications. The final results generated by these applications may therefore always be suspect. Moreover the implementation of such an environment will raise serious management issues,
for e.g. ownership of information and applications, obligations for information provision, application maintenance and version control etc.

A possible solution to the above mentioned problems lies in developing a suitable data warehouse to fulfil different information requirements [Inm96]. A data warehouse is defined as a subject oriented, integrated, non-volatile and time variant collection of data used to support decision making process [Inm96]. A data warehouse is a centralised repository of consistent data and information at various levels of granularity.

1.8 Objectives
This thesis attempts to investigate the reusability of information derived from primitive network monitoring data so that the required information can be generated efficiently. The concepts discussed in this thesis can be applied more generally to the management and processing of derived information in data warehouses and information systems.

Reusability of information is attempted by pre-processing the primitive data so that it can fulfil multiple information requirements efficiently in the context of the overall domain. The information resulting from this pre-processing operation is termed Intermediate Information. Intermediate Information elements are generated at the finest possible level of granularity and are cached in an Intermediate Information Base (Figure 1-13).

![Diagram](image)

Figure 1-13: Information Processing Based on Intermediate Information

These finest granularity Intermediate Information elements can be combined appropriately to generate Intermediate Information elements at the required levels of granularity. All applications for which the information requirements can be
fulfilled from Intermediate Information, retrieve the required Intermediate Information elements from the Intermediate Information Base. These applications may then derive the required information from these Intermediate Information elements. As data requests to the primitive database are reduced, the performance of the information system is expected to improve.

Chapter 2 of this thesis introduces data warehouses. Different issues involved in developing a data warehouse for network performance information are also discussed.

Chapter 3 explains the concepts of Intermediate Information and discusses their application in processing packet delay measurements. A model for information systems employing Intermediate Information is introduced. This chapter also discusses two data structures that can be used as Intermediate Information structures for packet delay information. The performance of information subsystems employing these Intermediate Information structures are compared.

Chapter 4 discusses different possibilities for dynamically selecting Intermediate Information structures in order to efficiently process packet delay information. A prototype information system employing these techniques selects an initial query execution plan on the basis of the user query specification. A query execution plan defines the Intermediate Information structure used by the system to process the packet delay information. As the system proceeds to derive the required information, the initial query execution plan is refined at each step in order to provide the required information as efficiently as possible. The process of query plan refinement is based on the number of Intermediate Information elements available to derive the required information.

Management and processing of packet loss and duplication summaries are described in chapter 5. These summaries are also derived at the finest levels of granularity. Appropriate finest granularity packet loss and duplication summaries may simply be added together to derive these summaries at coarser levels of granularity.

Chapter 6 demonstrates the construction of a prototype network performance information system from the components described in the previous chapters. This information system has been implemented as a simple single threaded server. This server can provide the network performance information as primitive data,
Intermediate Information elements with corresponding packet loss and duplication summaries or processed results with corresponding packet loss and duplication summaries. The ability of the server to provide these different types of information elements allows a variety of client applications to access the required information.

All prototypes and components explained in this thesis have been developed in Java using the object oriented design notation described in appendix A. These systems operated on a Sun UltraSparc Workstation under the Solaris operating system. In order to measure the performance of these components, the primitive data were generated by transmitting one test packet per second per test for two tests. Thus in one hour of monitoring, 1800 test packets were transmitted for each test. Packet delay information derived for these tests consisted of average delay values for successive 5% fastest test packets for all window sizes that are integer fractions of 24.
Chapter 2

Data Warehousing and Application to Network Monitoring Data

2.1 Introduction to Data Warehouses

Hammer et al describe a data warehouse as a repository of integrated information available for querying and analysis. As relevant information becomes available or is modified at the source, this information is extracted, translated to a common model (for e.g. the relational model) and integrated with existing data at the warehouse [HamGW95].

Data warehouses have been developed for query processing as opposed to transaction processing in operational or OLTP (On Line Transaction Processing) systems. Data within a data warehouse are organised in order to provide efficient access to processing applications that aid in the corporate decision making process.

Inmon identifies following as the basic features of a data warehouse [Inm96]

- Data warehouses are oriented towards the major subject areas of the corporation that have been defined in the data model
- Data in data warehouses are integrated by overcoming the inconsistencies in encoding data. When data are moved from the operational environment to the data warehouse environment, consistent coding convention is assumed.
- Data warehouses are time variant as they contain historical data (for e.g. 5 to 10 years old). These data are used for comparisons, trend analysis and forecasting. These data are not updated.
- Data warehouses are non-volatile as data are not updated or changed in any way once they enter the data warehouse, but are only loaded and accessed.

Data warehouses form part of a larger architectural environment as shown in figure 2-1 [Inm96]. The operational level of data holds the primitive data only and serves the high performance transaction processing community. The data warehouse holds primitive data from a number of heterogeneous sources. This primitive data is filtered, transformed and integrated seamlessly into the data warehouse. In a number of cases primitive data in the warehouse is also summarised at different levels of granularity.
The **departmental environment** contains information useful to different parochial departments of a company. The source of all departmental data is the data warehouse. The departmental level is sometimes called the **data mart** level or the **OLAP** (On Line Analytical Processing) level.

![Diagram of Architectural Environment for Decision Support Systems](image)

**Figure 2-1 : Architectural Environment for Decision Support Systems**

The final level of data in the architectural environment is the **individual level**. Individual data is usually temporary and small. At the individual level, much heuristic analysis is carried out. **EIS** (Executive Information Systems) processing is usually conducted at this level.

Thus at the data warehouse, queries can be answered and data analysis performed quickly and efficiently as the model and semantic differences are already resolved. It is for this reason warehousing is considered as an **active** or **eager** approach to information integration as compared to more traditional **passive** approaches where processing and integration initiates when a query is executed [Wid95], [HamGW95].

Warehousing is considered appropriate for [Wid95]

- clients requiring specific predictable portions of available information
- clients requiring high query performance but not necessarily requiring the most recent state of information
- environments where native applications at the information sources require high performance and large multisource queries are executed at the warehouse instead
• clients wanting access to private copies of information so that it can be modified, annotated, summarised etc.
• clients wanting to save information not maintained at the source for e.g. historical information.

Hammer et al suggest that warehousing is inappropriate either when absolutely current data is required or when clients have extremely unpredictable requirements [HamGW95]. Thus warehousing should be considered as a complement rather than a replacement of passive query processing schemes.

This chapter aims at reviewing principles of data warehousing. This chapter proceeds with explaining an architecture for data warehouses and a data warehousing system. Various issues in design and implementation of data warehouses are briefly discussed. Some issues in warehousing scientific data are also presented. Possible application of these principles with respect to network monitoring data are considered before concluding this chapter.

2.2 An Architecture for a Warehouse System
A complete warehouse based DSS (Decision Support System) may consist of a warehouse and a warehousing system (figure 2-2). The data warehousing system extracts data from the sources of information and integrates it into the data warehouse. The following paragraphs describe an architecture for a data warehouse and a warehousing system.

![Figure 2-2: Warehouse Based Decision Support System](image)

2.2.1 An Architecture for a Data Warehouse
Inmon describes an architecture of a basic data warehouse (figure 2-3) and explains different components of a data warehouse. He also discusses issues related to data warehouse design and implementation [Inm96].
Current detail data are considered to be of central importance as these reflect the most recent happenings which are usually most interesting. These data are voluminous as they are stored at the highest granularity. These data are usually stored on disks or other high performance media. Thus access to current detail data is fast but storage is expensive and difficult to manage.

Older detail data also form a part of the data warehouse. As these are expected to be accessed infrequently, slow, inexpensive and sequential access devices are used for storing older detail data. Storage is performed at a level of detail consistent with current detail data.

Lightly summarised data are distilled from the current detail data. Summarisation is generally conducted over a period of time for certain attributes of detail data.

Highly summarised data are maintained at the lowest granularity. These data are generated from lightly summarised data or from current detail. These data are also stored on disks and are easily accessible.

Metadata form an important component of a data warehouse. Metadata in a data warehouse are generally responsible for providing the following information.
Data Warehousing and Application to Network Monitoring Data

- **Data Location.** Metadata can include a directory to help the DSS analyst locate the contents of the data warehouse.

- **Mapping.** As data are transferred from the operational environment to the data warehouse environment, significant transformation takes place. These transformations may include filtering, projection, summarisation and aggregation of data. Metadata can provide a guide to the mapping of data as data are transferred from operational environment to the warehouse environment.

- **Process Information.** Metadata also provide information regarding the algorithms used for summarising the current detail data to the lightly and highly summarised data.

- **Structural Information.** Data in the data warehouse exist for a lengthy time span (for e.g. 5 to 10 years). Any data management system would normally change its structure within such a time period. Metadata can therefore be used to keep a track of and provide information on the structure of data as maintained by the data warehouse.

### 2.2.2 An Architecture for a Data Warehousing System

Widom has described an architecture of a data warehousing system [Wid95] (figure 2-4). *Information sources* are heterogeneous databases, knowledge bases, HTML documents and flat files etc. *Wrappers* translate information from the native format of the source into the format and data model used by the warehouse. *Monitors* are responsible for automatically detecting changes in data of interest in the source data and reporting them to the *integrator*.

![Figure 2-4: Architecture of a Data Warehousing System](image-url)
When any change is detected at the lowest (information source) level, for e.g. addition of a new source, change of data in the existing source, new or modified data, it is propagated to the integrator. The integrator is then responsible for installing information in the data warehouse. This installation may include significant transformation, for e.g. filtering, summarisation, format conversion etc. Integration may also involve extracting further information from same or different information sources.

A data warehouse may employ a customised or off the shelf database management system. Although logically a data warehouse is conceived as a centralised entity, it may be implemented as a distributed/parallel database system to improve performance. Propagation of information from the sources to the warehouse is generally performed as a batch process.

2.3 A Multidimensional Data Model for Data Warehouse
A multidimensional data model has largely been recommended as a suitable data model for data warehouses. The multidimensional data model is a technique for conceptualising business models as a set of measures described by ordinary facets of business. It is particularly useful for sifting, summarising and arranging data to facilitate analysis [Rad96]. Multidimensional data modelling uses facts, dimensions, hierarchies and sparsity.

Facts and dimensions are represented as tables in a relational database. In the star schema (figure 2-5) each dimension is described by its own table and facts are arranged in a single large table, indexed by a multipart key made up of individual keys of each dimension. Not all facts share the same dimensionality and multifact tables may be used.

Star schema generally segregate the numeric data from non-numeric data. It is strongly recommended that fact tables contain additive numeric data [Kim97]. Queries can then be designed to use the dimension tables for counts, control breaks, aggregation paths and searching for property elements. Many queries can be resolved without even touching the fact table [Rad96]. When facts are needed, key values are gathered from dimension tables and matching records are pulled from fact tables avoiding the costly joins and scans.

Attributes and sparsity are not entities and are not represented as tables. Attributes are extended descriptions and hierarchies of dimensions. Sparsity is handled
implicitly i.e. by simply not inserting records where these combinations are not valid.

Figure 2-5: Multidimensional Data Model

Inmon suggests that the star schema is not the sole data model for a data warehouse [Inm96]. Star schema applies as a design foundation to very large entities that exist in the data warehouse. Data models like entity relationship models apply as design foundations to the non voluminous entities found in data warehouse.

2.4 Issues in Data Warehouse Design and Development

Significant research has been conducted on various issues related to the design and implementation of data warehouses. A brief description of some of these issues is attempted here

2.4.1 Granularity

*Granularity* refers to the level of detail or summarisation held in the units of data in the data warehouse. Lambart discusses the issue of granularity in the context of *primitive* and *derived* data. A primitive data element describes an individual object or event whereas a derived data element describes many different objects or events. A primitive data element can only be measured whereas a derived data element is calculated from primitive or other derived elements [Lam96].
Data Warehousing and Application to Network Monitoring Data

A data warehouse designer tends to maintain multiple levels of granularity starting from primitive data. These multiple levels of granularity are generally maintained by generating different summaries that may be used to answer various queries by the warehouse users. Detailed data are included as most summaries limit the user’s ability to develop different summaries [Lam96]. Pre-calculating various summaries may substantially reduce the time taken by the warehouse to answer different queries. However, it may be very difficult to anticipate every possible information requirement of the warehouse user. Modelling all possible derived data as compared to all possible primitive data may also be very difficult. It is therefore mandatory that the warehouse must maintain the primitive data.

Inmon discusses two data structures commonly used for summaries in data warehouses [Inm96]. Of these, the *simple cumulative summary* is the simplest. Data entering the warehouse is summarised for a particular time window, for e.g. a day or a week etc. (figure 2-6).

An alternative is the *rolling summary*. In this, summaries calculated on a daily basis are combined at the end of the week to generate summaries on the weekly basis. Similarly summaries generated on a weekly basis are combined at the end of month to generate a monthly summary for that month (figure 2-7). Rolling summaries can only be generated for aggregates. Moreover rolling summaries incorporate some loss of detail. However they tend to be very compact and as the data gets older, less detail is maintained [Inm96].

![Diagram of Simple Cumulative Summary](image)
Simple cumulative summaries, in contrast, require more storage and significantly greater amounts of processing to generate the required information. However simple cumulative data do not incorporate as much loss as the corresponding rolling summary.

Inmon suggests that derived data and summaries are redundant [Inm88]. Moreover when compared with primitive data processing, derived data processing is based on algorithmically dynamic use of data. Thus in addition to striking a balance between the amount and type of summaries maintained by the system, there is a strong need to investigate reusability in data and information structures. This would allow generation of a different summary from a set of pre-calculated summaries without having the need to access primitive data. It is possible that there may not exist a data or information structure that is completely reusable, but it may be possible to define various levels or grades of reusability where information extracted from a specific instance of a data structure may have to be re-cycled to generate different information.

2.4.2 Purging Warehouse Data
Detailed data maintained for every subject in a data warehouse have a certain life span after which these need to be re-organised. Suitable summaries are generated for that specific period and the detailed data are purged. In some organisations, older detail data are transferred from the high performance medium i.e. disks, to a bulk medium, for e.g. magnetic tapes. Only summaries
of older data are maintained on the high performance medium as these are generally most frequently accessed.

2.4.3 Change Detection, Translation and Integration
The original sources of data need to be monitored for detecting changes in data that are relevant to the warehouse. These changes are then to be translated and propagated to the warehouse.

Widom suggests that changes in the information sources may be ignored and the entire warehouse may be re-computed periodically from complete sets of data passed from the information sources [Wid95]. This approach may be acceptable in scenarios where it is not important for warehouse data to be current and it is acceptable for the warehouse to be off-line occasionally [Wid95]. However if currency, efficiency and continuous access from the warehouse are required, then change detection, propagation and incremental folding of changes to the warehouse are preferred. Change detection can be accomplished with the assistance of the information sources. Widom also describes various types of information sources that can assist in change detection [Wid95].

Integrating data from the sources into the warehouse also represents serious challenges. These may be due to the fact that the structure of the warehouse is generally more complicated than that of its contributing sources. Moreover, information entering the warehouse may need to be integrated with information from other sources. Primitive data from the sources also need to be aggregated and summarised and these summaries are to be stored efficiently.

2.4.4 Cyclicity of Data
This issue is closely related to reporting of change detection. Cyclicity of data refers to the length of time a change of data in the operational environment takes to be reflected in the data warehouse. Inmon suggests introducing a wrinkle of time in the data i.e. to incorporate changes after a certain amount of time period. This wrinkle of time largely depends upon the type of data and the nature of application [Inm96].

2.4.5 Performance Optimisations
A data warehouse, if not architectured appropriately, may introduce significant performance bottlenecks. This is because of the amount of data that a data warehouse is expected to manage as well as the number of users attempting to
access that data. A number of techniques, ranging from defining suitable data structures to implementation of the warehouse on specialised hardware have been explored.

One of the most common techniques to improve performance is selective incorporation of redundancies and denormalisation [Inm96]. A normalised schema has a minimal redundancy, which requires that the value of no attribute of a database instance is replicated except where tuples are linked by foreign keys, i.e. the inclusion of the key of one relation in another [RyaSmi95].

Denormalisation is a set of database design techniques which aim to improve the performance of enquiries by creating redundant data. Kirkwood describes the following techniques in denormalising [Kir93]

- **duplicated data** where individual fields are introduced redundantly to reduce the number of records which need to be accessed in an enquiry.
- **derived data** where summary or calculated fields are introduced redundantly to reduce the number of records involved in the calculation.
- **surrogate keys** where artificial keys are introduced in place of larger, less efficient key fields.
- **vector data** where the concept of multiple repeating fields is re-introduced to group data in one record.

Performance can be further enhanced by indexing the warehouse appropriately. Richards discusses some advanced indexing techniques, for e.g. inverted and bit mapped indexes, along with the application of specialised hardware to improve warehouse performance [RicWWW].

**Partitioning** is a quite popular approach to enhance warehouse performance. A partition of a set of objects is a collection of subsets such that

- the union of all the subsets in the collection is equal to the original set
- the intersection of all the subsets in the collection is empty
- no subset in the collection is empty

This means that every element of the original set belongs to one and only one subset in the partition.

Partitioned data can be independently handled, thereby allowing parallelism or concurrency in data analysis. However Richards identifies a disadvantage of
partitioning in that it requires constant tuning in a dynamic environment where unpredictable queries are made. A query at any instant in time may require data from a specific partition but at another instant in time may require data from a number of partitions [RicWWW].

Other performance enhancement techniques may include provision of data marts, OLAP (On Line Analytical Processing) and query regulation techniques [RicWWW].

2.4.6 Provisioning of Auxiliary Data
Auxiliary data in an organisation may be defined as data that are not generated by the normal business process but are required for explanation of the results of analysis and queries. These would include reference tables and context information.

Reference data have to be stored in the warehouse on a historical basis. Thus in case past data from the warehouse are analysed, suitable reference data for the period of analysis will exist to supplement the result of analysis.

Context information represents other relevant factors that affect business transactions. Contents of the analysis of historical information without appropriate contextual information may provide misleading information. Inmon describes the following three levels of contextual information [Inm96]

2.4.6.1 Simple Contextual Information
This information relates to the basic structure of the data itself. It may include the structure, encoding, naming conventions and other metrics related to the data. Simple contextual information has been managed by dictionaries, directories and system monitors etc.

2.4.6.2 Complex Contextual Information
This describes the same data as the simple contextual information. However it addresses the higher level information aspects of the data, for e.g. product definition, marketing territories, pricing, packaging etc.

2.4.6.3 External Contextual Information
This information exists outside the organisation and plays an important role in understanding the organisational data over time. It may include
information like economic forecasts, political and competitive information, technological advancements etc. External contextual information can effectively describe the universe in which the organisation exists.

2.5 Distributed Data Warehouses

As mentioned previously, data warehouses are generally maintained as centralised data repositories on a location where an integrated view of the organisational data is required. One of the main reasons for centralisation of data warehouses is efficiency as attempting to integrate data from multiple sites may be inefficient [Inm96]. However under certain circumstances there is a need to distribute data warehouses appropriately.

The following paragraphs describe some of the situations where distributed data warehouses may be appropriate

2.5.1 Requirements for Local Processing

A data warehouse may be implemented as a set of distributed data warehouses if there is a requirement for significant processing at the local sites of an organisation (Figure 2-8). Local sites are autonomous for operational processing. Only on occasions and for certain types of processing are data sent to a central site. However a distributed data warehouse may not be a binary proposition and degrees of distributed data warehouses may exist [Inm96].

The local data warehouse contains data that are of interest only at the local level or for a specific community. A local data warehouse contains data that are historical in nature and is integrated within the local site. There is no coordination of data or structure of data from one local data warehouse to another.

The scope of the global data warehouse is the entire corporation. The data in the global data warehouse are integrated across the entire enterprise. This is accomplished by mapping data from the local operational systems to the data structure of the global data warehouse (Figure 2-9).

One of the major issues in a distributed data warehouse for geographically distributed organisations is the data access policy. As a principle, local data should be used locally and global data should be used globally.
Another issue relates to the routing of requests for information in the data warehouse environment, i.e. if the requests are being routed to the appropriate local or global data warehouse.

2.5.2 Economics of Implementation
The geographically distributed operation of an organisation may not be the only motivation for developing a distributed data warehouse. The designers of the data warehouse may want to implement it over distributed technology. This approach may have a number of advantages. For example, the cost of hardware
and software for a data warehouse when initially loaded onto distributed technology is much less than if the data warehouse were initially loaded onto classical, large, centralised hardware.

Moreover, if distributed technology is used, there is no theoretical limit on the amount of data that can be placed in the warehouse. If the volume of data inside the warehouse begins to exceed the limit of a distributed processor then another processor can be added (Figure 2-10). This progression of adding data continues in an unimpeded manner [Inm96].

However, after a few servers have been added to the data warehouse, the traffic on the network may be extensive. Moreover compiling voluminous results of queries from multiple servers may also be a complicated process.

![Figure 2-10: Expansion in a Data Warehouse Implemented on Distributed Technology](image)

2.5.3 Warehousing Data of Disjoint Activities
It may be appropriate to develop distributed data warehouses where different activities of an organisation are not related to one another. Data warehouses may be developed for each individual activity and may be implemented over distributed technology.

2.5.4 Data Marts
Distribution can also be achieved in an information processing environment by employing several data marts which are fed by centralised data warehouses.
Data marts were developed in the early 1990's to overcome various performance and structural problems faced in operating and using massive data warehouses [Dem94].

Data marts can be defined as community specific data stores focusing on Decision Support Systems' (DSS) end user requirements. Thus data marts attempt to solve the enterprise DSS problem by presenting only the data that an end user constituency requires in the form close to the constituency's business model [Dem94].

Approaches based on pure data marting address the decision support needs of only small companies with few knowledge workers, single markets and simple product lines. They cannot meet the needs of large organisations with many distinct knowledge worker communities, many products and markets and constant reorganisation in response to market conditions [Dem94]. Richards, in his white paper, suggests that data marts can lead to islands of data warehouses where the information services need to build bridges between these data marts [RicWWW].

However, as the data warehouses are populated, an increasing amount of DSS activity is carried out in the warehouse. Moreover DSS analysts find it more difficult to customise data inside the warehouse. They also have to rely only on the analysis software that is available in the warehouse [InmWWW].

Alternatively an organisation can manage a suitable data warehouse and provide the departmental DSS analysts appropriate data marts. In this way data related to specific departments is fed to the departmental data marts from the data warehouse. The data marts appropriately structure and summarise the granular data provided by the data warehouses.

The departmental DSS analysts can customise the data in the data marts according to their requirements. The amount of historical data required is only a function of the department and not the whole organisation. This results in expeditious processing of queries. Moreover the departmental DSS activities do not have any impact on the operation of the warehouse. Departmental DSS analysts can, therefore, install appropriate analysis software on their data marts.
Inmon describes a number of issues that need to be dealt with while developing a data mart. Some of these may be listed as follows [InmWWW]

2.5.4.1 Data Loading
A principle issue concerns the loading of the data marts. Data can be pushed into the appropriate data marts by the data warehouse. Alternatively, the data marts may demand (pull) the required data from the data warehouses. The frequency of loading, customisation and summarisation of data, creation of suitable metadata as well as the integrity of data and related information need to be considered by the data mart designer.

2.5.4.2 Data Model
The requirement of an appropriate data model in the data mart depends upon the size and formality of the data mart, type of database management systems used (relational vs. multidimensional) as well as the type of data maintained (primitive vs. summary data).

2.5.4.3 Capacity Management
The data marts have to be periodically purged. The removed data may be deleted permanently, archived or condensed. A suitable purging policy must control this operation. This policy should dictate the amount of data to be purged, purging criteria and the disposal of purged data.

2.5.4.4 External Data
External data required by more than one data marts should be placed in the data warehouse. The data warehouse then provides these external data to the appropriate data marts. This ensures that external data are procured only once and that redundancy of external data is controlled. Another related issue is the storage and management of pedigree of external data (i.e., source of external data, date and amount of acquisition, data description and usage criteria).

2.5.4.5 Performance
Inmon describes various performance issues that may be encountered while designing data marts. He lists some techniques that may enhance the performance of data marts. These range from suitably structuring and
referencing the data to pre-calculating the required information [InmWWW].

2.5.4.6 Security
Data in a data mart may contain sensitive information and there may be a requirement to ensure that these data are maintained securely. The level of the security techniques employed depend largely upon the sensitivity of the information maintained. Various approaches to security may be employed, e.g., firewalls, log on/log off, encryption etc.

The data marts may need to be continuously monitored. Monitoring is generally required to track the usage of data as well as its content. Usage tracking involves determining the data being accessed, response times, amount of data being requested as well as peak operating hours. Content tracking attempts to determine the actual contents of the data, corrupt data in the data mart, rate of growth of the data mart etc.

2.5.5 Data Warehouse on Internet/Intranet
With the rapid emergence of the Internet, there is a strong inclination to move the functionality of databases and data warehouses towards the servers. It is believed that this trend allows for much easier configuration of the desktop; the minimal system would need only a web browser [Zag97].

The administration of the entire system is centralised at the servers (i.e. across fewer machines). Moreover the Internet provides complete insensitivity of the end user to the operating environment of the server systems. As a result, the end users can easily configure the services they need over suitable platforms.

In spite of the above mentioned advantages, this model may have the following short comings [Zag97]
- scalability issues become of paramount importance
- bandwidth of the networks can become a serious inhibitor as large amounts of data will flow
- users of the mobile computing systems need the capability to be productive even when disconnected

These needs will continue to tempt the users to stick to the fat client machines.
2.6 Warehousing Scientific Data

Management of scientific data represents serious challenges for scientists and researchers. Much of scientific data are characterised by large volume, low update frequency and requirements for indefinite retention [FreJP90]. Additionally, scientific information structures are complex as they involve a large number of objects and associations between them [Sho93]. This complexity of the scientific information structures is generally independent of the volume of data.

Scientific data for a number of domains are being generated at an enormous rate [LetBer94] as a result of monitoring of different phenomena, experiments and simulations. Moreover these data are available to the researcher from a number of different sources. Various heterogeneous databases store these data at the source. Hansen et al suggests that scientific data sources may either be physically heterogeneous or logically heterogeneous [HanMai94]. The former addresses the issues of availability of scientific data from a wide variety of media whereas the latter arise due to the insuitability of common data models to support complex scientific data. Possibilities also exist of scientific data being erroneous, incomplete and inconsistent in representation and context [FreJP90]. This imposes a logistical barrier to the access of scientific data.

Access to scientific data of a particular dimension of science may be significantly facilitated by warehousing it. Adequate performance enhancements in data analysis can also be obtained by the physical integration of contributing source databases into a data warehouse [AbeHem96].

French defines scientific data as information, usually numeric, that has been derived from some measurements, observations or calculations. He further defines a scientific database as an organised collection of scientific data on a well defined topic [Fre91]. A scientific data warehouse may therefore be defined as an integrated, time variant and non-volatile repository of scientific data. Data for a scientific data warehouse are contributed by multiple, potentially heterogeneous, scientific databases.

The characteristics of scientific data and the associated processing requirements present additional issues in constructing a suitable scientific data warehouse. Due to the complexity of scientific data generated in a number of scientific domains, it may not be feasible to store scientific datasets from all activities of a particular topic in a relational database [FreJP90]. Simple experimental data i.e. observations
have to be analysed in conjunction with experiment configuration and instrumentation data [Sho91] to provide accurate results. Once a particular dataset is integrated into the scientific data warehouse, all the experiment configuration and instrument data related to that dataset have to be integrated as well. Moreover data generated at various stages of the analysis process may have to be stored along with the relationships between these analysed datasets [Sho91].

Scientific databases are required to support extensive metadata. This includes information required to identify datasets of interest, their contents, validity and source of information [Fre91]. Without metadata, data in a data warehouse cannot be interpreted and is therefore worthless. A scientific data warehouse must maintain adequate metadata that provides, in addition to the above information, information regarding the mapping, summarisation and transformation processes and data structures used in constructing and populating that data warehouse.

Using relational databases to manage scientific data generated by various scientific processes may introduce certain inefficiencies. A number of different data models and database technologies have been proposed for the management of scientific data. These include object oriented databases, extended relational databases and extensible database management systems [Fre91], [Sho91]. Management of data in scientific data warehouses may also present various issues in terms of choice of specific data models. Different summaries and pre-processed information may not be supported efficiently by the data model used for detailed data. It may be appropriate to use layered database technology as proposed by Shoshani [Sho93]. These layered architectures may manage different data files and data models at the lowest level and provide information to the applications as semantically rich objects generated by the higher level layers.

2.7 Warehousing Network Monitoring Data

Network performance monitors can generate significantly large amounts of data by measuring different parameters of the network elements. Network service providers need to analyse network performance data in order to determine whether the service is being provided to the subscribers in compliance with the service level agreements (SLA). Subsequent analysis conducted on the monitored data may vary from simple statistical analysis such as calculation of various means and aggregates to certain advanced information processing techniques. These techniques attempt to relate the semantics into the information thereby leading to knowledge
representation and symbolic processing that results in the capability to perform significant reasoning with the data [MamSmi91].

Information processing and analysis is based on the careful study of the nature of data along with appropriate organisation and management of primitive data and information that is derived from it. Thus there exists a need to appropriately organise, manage and pre-process the monitoring data so that the querying and analysis requirements of the network operators may be satisfied efficiently.

Network monitoring systems may be based on an elaborate architecture. They may consist of numerous distributed monitoring stations, that may be making performance measurements on different network components. There exists a need to collect, organise and integrate the measurements made by these monitoring stations in a suitable manner so as to assist efficient processing. The following paragraphs present some issues related to warehousing network performance data.

2.7.1 Management of Detailed Network Performance Data
Detailed network performance data obtained by intrusively monitoring a network may be conveniently stored as a set of relational tables. This data may however have to be appropriately partitioned to enhance the performance of the data warehouse. A simple partitioning scheme would store all the data for a particular test in a separate file for each month. These files are located in different directories where each directory represents a particular month of the year (figure 2-11). Thus the sub-directory 11 in the directory 1996 will contain files of network measurements for the month of November in 1996.

2.7.2 Management of Monitoring Experiment Configuration Data
Network monitoring operations may be conducted by transmitting different test packet sizes and types from a set of source test stations to a set of destination test stations. Test packets of a specific type or size transmitted on a specific link (i.e. from a specific source test station to a specific destination test station) represent a particular test or a monitoring experiment. Each test may be represented by a unique test identifier (test ID), which provides information regarding the transmitting and receiving test stations as well as the type or the size of the test packets used in that test.

A test identifier thus represents the configuration of a particular network monitoring experiment. Details of the data elements that constitute
configuration data need to be managed appropriately within the data warehouse. Figure 2-12 shows one of the possible storage strategies for these data.

The **monitoring station sites** (i.e. source and destination test stations) active during a particular month are listed in the **monitoring stations table**. Similarly the **packet sizes** used for these tests are also listed in the **packet size table**.

If

\[
S = \text{Set of all active test station sites for a month}\quad \quad \quad \quad [2.1]
\]
then

\[ S \times S = S^2 = \text{Set of ordered pairs representing all possible routes for a month} \]  
[2.2]

Similarly if

\[ P = \text{Set of all packet sizes used during the month} \]  
[2.3]

then

\[ S \times S \times P = S^2 \times P = \text{Set of ordered triplets representing all possible tests conducted during the month} \]  
[2.4]

However in an actual testing scenario, some of the routes may not be tested. Similarly some of the packet sizes used during the monitoring session may be used to test some specific routes. This would give rise to some invalid routes as well as some invalid tests. These may be specified by the system operator and are also maintained in separate tables (figure 2-12).

If

\[ IR = \text{Set of ordered pairs representing all invalid routes} \]  
[2.5]

and

\[ R = \text{Set of ordered pairs representing all valid routes} \]  
[2.6]

then

\[ R = S^2 - IR \]  
[2.7]

Similarly if

\[ IT = \text{Set of ordered triplets representing all invalid tests} \]  
[2.8]

and
\[ T = \text{Set of ordered triplets representing all valid tests} \quad [2.9] \]

then

\[ T = S^2 \times P - IT \quad [2.10] \]

Alternately the warehouse may only maintain a table of all valid tests conducted in a month along with the monitoring station table and packet size table (figure 2-13).

<table>
<thead>
<tr>
<th>Size</th>
<th>Site Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM</td>
<td>0</td>
</tr>
<tr>
<td>BS</td>
<td>1</td>
</tr>
<tr>
<td>EH</td>
<td>2</td>
</tr>
<tr>
<td>LN</td>
<td>3</td>
</tr>
<tr>
<td>MR</td>
<td>4</td>
</tr>
</tbody>
</table>

**Monitoring Stations Table**

<table>
<thead>
<tr>
<th>Packet Size</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 Bytes</td>
<td>0</td>
</tr>
<tr>
<td>700 Bytes</td>
<td>1</td>
</tr>
<tr>
<td>1500 Bytes</td>
<td>2</td>
</tr>
</tbody>
</table>

**Packet Size Table**

<table>
<thead>
<tr>
<th>Valid Tests</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM-BM-54</td>
<td>010</td>
</tr>
<tr>
<td>BS-LN-1500</td>
<td>132</td>
</tr>
<tr>
<td>BS-LM-64</td>
<td>100</td>
</tr>
<tr>
<td>LN-BS-1500</td>
<td>312</td>
</tr>
<tr>
<td>BM-LN-1500</td>
<td>012</td>
</tr>
<tr>
<td>LN-BM-1500</td>
<td>302</td>
</tr>
</tbody>
</table>

**Valid Tests Table**

2.7.3 **Management of Summaries and Pre-processed Information**

Queries posted to the warehouse would generally request the network behaviour for a particular time period. Different information may be required for different types of analysis, for e.g. some queries may request average delays for a certain time period within a specific window whereas others may require the calculation of a frequency distribution of delays for the specified time period.

It may be possible to pre-process the detailed data in a manner so that the resulting information structure may be used to provide answers to different queries. An ideal data structure containing the pre-processed data would

- provide maximum possible information for various information processing activities
- provide sufficient degree of **re-usability or re-cyclability** of data
• present reasonable requirements (data storage, computational expense etc.) for its above mentioned attributes.

The detailed data may be pre-processed and stored at different levels of granularity along with the summaries and the results of analysis. The warehouse management system may then attempt to access the pre-processed data in response to a particular query. As it is expected that pre-processed data will provide a certain degree of re-usability, in the absence of pre-processed data of the required granularity, it may be constructed from pre-processed data maintained at a finer level of granularity. If pre-processed data for a specific query do not exist at any level of granularity, then the detailed performance data are accessed to generate the required pre-processed data as well as the summaries and results. This pre-processed data may then be used for subsequent queries thereby enhancing warehouse performance.

2.7.4 Management of Auxiliary Data
Auxiliary data such as reference tables and context information can enhance the information generated by the information processing applications. Moreover these also provide an indication regarding the accuracy of results derived from a particular data set.

Two types of auxiliary data related to network monitoring are described here. The first type includes regular snapshots of network configuration and link status. Results of analysis for a particular period can be correlated with snapshots of network configuration and link status for that period. This can provide an explanation regarding events such as significant variation in delays, increase in network losses and other forms of abnormal behaviour.

In a distributed network monitoring system, a central control station must check the status of each monitoring station regularly and maintain a suitable log (i.e. instrumentation data). Various test station abnormalities, for e.g. loss of clock synchronisation or a disk crash resulting in a failure to record the monitoring data can generate inaccurate and inconclusive reports. The inaccuracies in these reports may not be detected unless appropriate test station status data is not included in the reports.

Thus a warehouse for network performance data must have the ability to store different context information that may be made available from different
solutions. Means of acquiring this information and its integration into the warehouse are important research issues.

2.8 Users of Information Systems and Data Warehouses

Ein-Dor & Segev [EinSeg78] provide a summary of the cognitive styles of different users of the information systems. They suggest that the interaction of the cognitive styles of users with the output of an information system has a significant bearing on its success. Cognitive styles can be decomposed into two dimensions, i.e. information gathering and information evaluation.

Along the information gathering dimension, individuals tend to be either preceptive or receptive. Preceptive individuals filter incoming data according to their prior concepts and tend to perceive complete pictures. Receptive individuals are sensitive to bits of data independently of their conceptual frameworks. Thus the more receptive people are attentive to details and look for complete data sets and avoid preconceptions. Those who are more preceptive look for hints, relationships in the data and extract parts of data to create new combinations and precepts.

The information evaluation or problem solving dimension is related to the sequence in which data that have been gathered are then analysed. On this dimension, individuals who are basically systematic look for a method that when used will guarantee the best solutions. Individuals who are more intuitive do not commit themselves to any particular method. They may try and retry many different approaches in a trail and error fashion. Intuitive thinkers tend to redefine a problem frequently, relate to the total problem rather than to its component parts, jump backwards and forwards in the process of analysis and simultaneously formulate, evaluate and abandon alternatives in rapid succession. On the other hand, systematic thinkers look for an approach and a method, and advance through the problem solving process in an orderly sequential manner.

Cognitive style is therefore the location of the individual in the two dimensional space formed by the styles of information gathering and information analysis. Problem solvers may be receptive-systematic, receptive-intuitive, perceptive-systematic and perceptive-intuitive. Usually they are somewhere along each of the dimensions and not necessarily at the theoretical extremes.

Glassey-Edelholm [Gla97] categorises the visitors of the data warehouse into the following categories
• **Report viewers** typically look for answers in the same location on the same report on a predictable schedule. They do not worry about the tools used to generate the required reports and typically have no tolerance for systems that requires them to remember how to do something. They may work with the system once a week or once a month and so their system must prompt them through even the simplest processes. Report viewers do not demand immediate response as long as the system ultimately delivers the required information.

• **Data tourists** require recommendations regarding interesting data elements. They want a system that can collect and cluster data together so that it can make sense. Data tourists typically prefer a structured report initially but tend to exploit any variability that they can find in these reports. They also need to customise their own reports and save a record of the paths they crossed in their search through the data. They are less tolerant than the report viewers and require a good response time during a single session.

• **Information surfers** are the most demanding and fortunately the smallest percentage of the average population. They constitute at most 10% of the typical user community. Surfers want the ability to ask any question of any data at any time with almost instantaneous response. Surfers are rarely satisfied by the existing reporting systems as they always ask unexpected questions. They need true interactive multi-dimensional analysis and will be frustrated by the restrictions of tightly controlled query environments.

• **System planners** interact with data regularly during the development and operation of the system. Data access tools with thick static semantic layers are unsuitable for the planners. Planners are ultimately responsible for system support and therefore look at the end users tools in terms of the support burden.

Inmon [InmWWW] explains two categories into which the departmental DSS (Decision Support System) analysts, i.e. users of data marts, can be divided. One category is the **farmers**. DSS farmers know what they want and they predictably go to that place to find the required information. Another category is the **explorers**. DSS explorers do not know what they want. They look at the data in a random sporadic manner. There are generally more farmers at the data mart level than explorers. Thus the data mart environment has a very strong bias for farmers rather than explorers.

2.9 **Summary**

Data warehouses represent the architectural foundations of decision support systems [Inm96b]. Data warehouses extract various data maintained by an
organisation in potentially distributed and heterogeneous databases and integrate that information after resolving various inconsistencies in that data. Thus data warehouses provide a historical, integrated and subject oriented view of the data maintained by the entire organisation.

Data warehouses are designed to efficiently process queries and perform high level data analysis on the historical data. On the contrary OLTP systems are generally optimised to efficiently manage operational database transactions.

Data warehouses manage current detailed data, old detailed data, lightly and highly summarised data. Old detailed data are generally archived on slow and inexpensive media. Maintenance of data at multiple levels of granularity provide sufficient information to efficiently perform analysis. Lightly and highly summarised data may not be suitable for certain analysis requirements. In such cases analysis may directly be conducted on detailed data.

Additionally a data warehouse must maintain an effective metadata. Metadata provide information on the location of data in the data warehouse, mapping between operational and warehoused data, summarisation processes and the structural evolution of the warehouse. As analysis are conducted on historical data that are maintained for a significant amount of time, it is essential that various contextual information and reference data are also maintained on historical basis. This allows for the correct interpretation of analysed results.

Because of the amount of data managed by a warehouse, adequate attention needs to be paid on various performance issues. A number of techniques, ranging from application of specialised hardware to the use of appropriate indexes, are available to boost the performance of data warehouse.

Efforts have also been made to distribute the corporate data warehouses appropriately. Data warehouses may be distributed if significant amount of processing needs to be conducted at the local sites. Each site then maintains its own data warehouse in addition to the global warehouse maintained centrally. Data warehouses may be implemented using distributed technology because of different advantages offered by it. For example, the cost of hardware and software for a data warehouse when initially loaded onto a distributed technology is much less than if the data warehouse were initially loaded onto classical, large centralised hardware. Moreover distributed technology offers scalability, i.e. when
the data in the warehouse exceed the limits of one server, another server may be added. Data warehouses may be implemented over distributed technology when they are built to maintain information regarding disjoint activities of the organisation.

Distribution can also be achieved in an information processing environment by employing several data marts which are fed by the centralised data warehouses. Data marts are community specific data stores that focus on end user requirements. Significant performance enhancements may be expected if a suitable combination of data warehouses and data marts is employed to manage organisation's data.

Research has also been conducted into moving the functionality of the databases and data warehouses back to the servers and provide access to these via the Internet / Intranet. It has been suggested that the end users would only need a thin client, i.e. a minimal system with a web browser, which can be configured easily. This would also facilitate the administration of the centralised server system. However other issues related to loading of the servers, scalability and applicability to mobile environments is actively being researched.

Although data warehouses were initially developed for business and commercial query processing and decision support, their application in other domains, namely science and technology has been attempted. Numerous scientific processes generate enormous amounts of data that may be managed locally in heterogeneous databases. Attempting to analyse and correlate scientific information directly from these heterogeneous sources can prove to be inefficient. Performance enhancements in data analysis processes have been achieved by the physical integration of contributing source databases into a data warehouse. However as characteristics of scientific data and corresponding processing requirements are significantly different from data generated by business and commercial process, development of scientific data warehouses may present some serious issues.

Activities aimed at monitoring data communication networks generate significant amount of data that need to be appropriately managed to facilitate analysis. This can be accomplished by developing a suitable warehouse to manage the network monitoring data collected by a number of distributed network monitoring stations. In addition to the monitoring data, auxiliary information such as monitoring stations' status, status of network elements and network topology may also required
to be stored. This allows correlation of the results of the analysis with auxiliary information to assist in the interpretation of these results.
Chapter 3

The Application of Intermediate Information Concepts to Process Packet Delay Measurements

3.1 Introduction

Information requirements for many processes are generally not known completely during the analysis and design phase of an information system development project [Mar84], [Lan85], [WatHR97]. This poses a major hurdle in the development of such systems. Moreover it has also been argued that information systems generally focus on the demands of the users rather than their needs. In fact only the expressed needs of the users are focused on whereas these expressed needs may only be a distant approximation of actual needs [LanRap81]. It is quite common that once an information system has been used for sometime, users initiate further requirements for information.

This can result in the major restructuring of several components of an information system. Moreover, it is difficult to model all the possible types of summary data for a particular application that may be derived from the primitive data sets [Lam96]. It might also not be feasible to pre-calculate all the possible summaries and store these for subsequent analysis. This may result in the management of extensive summary data.

Thus every time a new set of information is required, the entire information system or parts of it may need to be re-engineered significantly. This adds to the costs of developing and maintaining information systems. Tremendous pressure builds up on data analysts to provide the required information in time so that suitable business decisions may be taken. Such ad hoc developments ultimately result in an unstructured information and data analysis system.

Most business and scientific processes generate numeric data. These numeric data are generally summarised to derive the required information elements that have been specified during the system analysis phase. These information elements are generally those that may suitably fulfil the routine information requirements of the organisation. In certain cases these summaries are additive in nature (for e.g. counts, sums etc.) thereby allowing derivation of further summaries without having to access the primitive database. In certain other cases, summaries may not be additive in nature. In such cases further summaries may need to be derived.
from the primitive database. Frequent access to the primitive database reduces the efficiency of the information processing applications.

It may therefore be desirable to structure the information derived from the primitive data in a manner so as to enhance its reusability and recyclability. Information has been regarded as one of the essential resources of an organisation (the literature here merges the boundaries between data and information) [GruSon83]. It has been suggested that information has all the properties of a resource [CCTA90],

- it is essential as all activities are dependant on it
- it has a cost as significant effort is spent on its collection, dissemination, processing, storage, archiving and disposal
- it is used to assist in achieving the aims and objectives of the organisation

It is for these reasons that significant effort is spent to efficiently acquire, store, process and disseminate information.

It has also been suggested that like all other production factors of an organisation, information also has alternate usage and value. Information that has been collected to fulfil a particular information requirement may be reused or recycled (re-processed) with other information elements to fulfil other information requirements [LanKen87].

Core data analysis operations like data mining typically represent only a small fraction of the overall processing requirement [BraKK96]. Most of the effort is expended in data acquisition, pre-processing and transformation as well as different post-processing activities. It is therefore desirable to pre-process the primitive data to generate Intermediate Information that may be used in the subsequent processing to derive the required information. An Intermediate Information structure is expected to provide maximum possible information, in the context of the application, without forcing the application to access the primitive data. Moreover it may also be desirable to generate Intermediate Information with coarser granularity by combining appropriate Intermediate Information elements derived at a finer granularity. Intermediate Information elements may also be cached for subsequent reuse. As accesses to the primitive databases are expected to reduce, the efficiency of the information processing activities will increase significantly.
This chapter attempts to describe the basic concepts of Intermediate Information and its application in maintenance and processing of network performance data. A model for information systems based on Intermediate Information is also described. Two information structures for network performance information, i.e. packet delays sorted in increasing order (Sorted Delays) and frequency distribution of packet delay values (Delay Distribution) are explained. The performance of an information system using Sorted Delays is compared with the performance of an information system using Delay Distributions. The chapter then describes an object oriented architecture for a network performance information sub-system that employs Intermediate Information. In the end, the impact of information reusability on system performance is demonstrated.

3.2 Basic Concepts of Intermediate Information

A number of researchers have defined information as being refined, extracted or processed from raw data and has a meaning to the user requesting that information [Mea76], [Sta85]. Thus for numeric data, information may be derived by the application of mathematical and logical operations on a set of data elements. These operations may be simple, such as addition or they may involve extensive numerical computations such as calculation of a Fourier transform. Intermediate Information may be defined as a set of elements that are derived from the primitive data and may be used to derive other information elements.

If $S$ is a set of data elements that have been extracted from a database, then useful information $U$ may be defined as a set of values derived from $S$ by performing an operation $O$:

$$U = O(S) \quad [3.1]$$

It may be possible to generate some information $I$ from the primitive dataset $S$ by performing an operation $P$:

$$I = P(S) \quad [3.2]$$

such that it is possible to derive information $U$ by performing an operation $\theta$ on $I$:

$$U = \theta(I) = \theta(P(S)) \quad [3.3]$$
The Application of Intermediate Information Concepts to Process Packet Delay Information

$I$ may therefore be termed as Intermediate Information as it is generated by performing the operation $P$ on $S$, but it can also generate $U$ if operation $\theta$ is applied to it.

The following properties can be considered necessary in defining an Intermediate Information structure for a particular application,

a. **Richness Property**

Intermediate Information structures may allow generation of $k$ different information elements if $k$ different operators are applied to them

$$U_k = \theta_k(I)$$  \[3.4\]

when $k > 1$.

b. **Combination Property**

If $S'$ are $m$ partitions of the primitive dataset $D$ i.e. $\bigcup_i S'_i = D$ and $\bigcap_i S'_i = \phi$, and there exist Intermediate Information $I'$ for each partition, then it should be possible to generate Intermediate Information $I$ for $k$ partitions from the Intermediate Information pre-processed from $m$ partitions by performing a combination operation $C$. $k$ is an integer fraction of $m$ and:

$$S_i = \bigcup_y S_i^{\frac{m-n_{S_i}}{k}}, 1 \leq l \leq k, 1 \leq y \leq \frac{m}{k}$$

$$I_i = C(I_i^{\frac{m-n_{S_i}}{k}}) = P(S_i)$$  \[3.5\]

The combination operation $C$ may vary in complexity depending upon the nature of the Intermediate Information.

c. **Complexity Property**

If $U = O(S) = \theta(I)$ and $\xi$ represents a function to determine the complexity of an operation, then $O$ should be more complex than $\theta$:

$$\xi(\theta(I)) < \xi(O(S))$$  \[3.6\]

These properties suggest that Intermediate Information should be able to fulfill multiple information requirements. Different instantiations of an Intermediate
Information structure at a finer granularity should be able to generate Intermediate Information at a coarser granularity. Operations extracting useful information from Intermediate Information should be more efficient than ones that derive the same useful information from the primitive database.

The accuracy of the final information derived from such Intermediate Information may also be an important criteria for the selection of a particular Intermediate Information structure.

Thus the structure, as well as the content, of Intermediate Information depends largely on the final information processing requirements. These also depend upon the characteristics of the processing platform. For example, insufficient disk space to cache a more comprehensive Intermediate Information structure may force generation and storage of a more specific and thus smaller information structure.

This suggests that the selection of a suitable Intermediate Information structure may involve an extensive study of the current and the future information requirements of the users of the system as well as the phenomena being modelled.

3.3 A Model for Information Systems Based on Intermediate Information

A model for an information processing system is described here. This model relies heavily on *reusability* and *recyclability* of information that is processed within the environment.

Reuse may be defined as the adoption of an already created component in preference to the design and construction of a new component [CCTA94].

Reuse is generally considered to be of the following kinds [CCTA94]

- *tailoring reuse*: modifying a copy of an existing component
- *copy reuse*: using a copy of a component
- *true reuse*: using an existing implemented component (not a copy of it)

It is suggested that an information system development project can reuse a wide variety of components including code, specifications, designs, tools and well as data.
Information processing systems process the data generated by an activity or a process (business or scientific) and derive information that is required for subsequent analysis. The information derived at the output of the information processing system is analysed with information received from external sources (Figure 3-1). The results of these analysis are then used to control the data acquisition, information processing and analysis activities as well as the process so that it operates within the pre-defined limits [Inm88] [GruSon83].

![Information Processing Environment Diagram](image)

**Figure 3-1 : Information Processing Environment**

The Data Acquisition system may be instructed to vary the sampling rates or to sample other attributes of the process being monitored. The Information Processor may be instructed to perform different types of processing on the input data and provide various information elements at the output. Similarly the Information Analyser may be instructed to perform different analysis operations in addition to or instead of the analysis operations it usually performs.

Figure 3-2 shows an Information Processor that employs Intermediate Information and reuses it to fulfil different information requirements. The pre-processor processes the primitive data to generate Intermediate Information at the finest level of granularity and caches it in disk files as an Intermediate Information Base. Most of the information requirements are then fulfilled from the Intermediate Information Base and the accesses to the primitive database are reduced. If a particular information query requests information at a granularity coarser than the default finest granularity, then appropriate finest granularity Intermediate Information elements are combined to generate Intermediate Information at the
required level of granularity. The required information is then derived from this Intermediate Information element.

The design of the pre-processor represents a major issue in the development of the architecture shown in figure 3-2. Activities involved in developing a suitable pre-processor include choosing an Intermediate Information structure that retains significant information. Moreover, the Intermediate Information structure should possess characteristics that would allow its elements to be re-cycled to generate further Intermediate Information elements. The pre-processor must also employ some efficient Intermediate Information re-cycling algorithms.

Figure 3-2: Information Processor Employing Intermediate Information

The design of the Processing Elements depends largely on the Intermediate Information structure. The operation of the architecture shown in figure 3-2 may be represented by the state transition diagram shown in figure 3-3.

One of the applications of such a model may be to facilitate processing of historical data in data warehouses [Inm96]. Data warehouses are an application of eager or active approaches to data integration where the relevant data are extracted from the sources, filtered and integrated as well as suitably summarised in advance of user queries [HamGW95], [Wid95].
Information is generally required from historical data for analysis at varying levels of granularity, for example, total daily sales of an item during a month, or average rainfall during each month of a year, etc.

For the model presented in figure 3-3, consider $T_h$ to be the time required to generate one finest granularity Intermediate Information element from the primitive data. Let $T_g$ represent the time required to generate a particular coarser granularity Intermediate Information element from the corresponding finest granularity Intermediate Information elements once these elements have been accessed. $T_r$ represents the time required to access a finest granularity Intermediate Information element that has been generated previously and stored for subsequent reuse. $T_q$ represents the time required to derive specific information elements from an Intermediate Information element at the required level of granularity. If $N$ finest granularity Intermediate Information elements are required to fulfil a particular information requirement, and $a$ such elements are available, then the response time $R$ required to answer the query may be represented as

$$R = aT_r + (N - a)T_h + T_g + T_q$$  [3.7]

Figure 3-3: Operating States of an Information Processor Employing Intermediate Information
Values of $T_r$, $T_h$, $T_q$ and $T_g$ may vary for different levels of granularity. These values may be determined by experimentally measuring the response times of the information system.

### 3.4 Employing Intermediate Information Structures for Packet Delay Information

Performance data generated by the network monitoring stations performing intrusive monitoring can be classified as

- data regarding the test packets transmitted by the transmitting test station
- data regarding the test packets received by the receiving test station

Each packet transmitted or received is represented by the following tuples:

$x = (pid, tid, size, t_x) \quad [3.8]$

$r = (pid, tid, size, t_x, t_r) \quad [3.9]$

where

- $x$ represents a record identifying a transmitted packet
- $r$ represents a record identifying a received packet
- $pid$ represents the packet identifier
- $tid$ represents the test identifier
- $t_x$ represents the time when the test packet is transmitted
- $t_r$ represents the time when the test packet is received

An element of a particular tuple will be represented by the `tuple[element]` notation. Thus the transmit time of a received test packet will be represented as $r[t_x]$. Similarly the test identifier of a transmitted test packet will be represented as $x[tid]$.

Monitoring data are stored in a database. The database consists of a pair of tables for each test identifier. One table contains details of the transmitted test packets, and the other contains details of received test packets.

As the transmit time of each transmitted test packet is different, it may be used, along with the packet identifier and test identifier, to uniquely identify each record representing a packet. Thus the transmit time may be used as keys for tables.
containing data for transmitted as well as received test packets. Moreover both tables are chronologically ordered. This allows the application of search methods (for e.g. binary search) to efficiently retrieve the required data sets from the database for analysis.

Monitoring data are considered to be the raw material for the information derivation processes with information being its output. Transformation of data into information may proceed through a number of different stages. These include pre-processing and transformation of primitive data, the derivation of suitable summaries from the pre-processed and transformed data as well as the association with appropriate qualitative data to suitably explain these summaries.

Information processing in a monitoring environment presents a dynamic scenario, as information requirements may not be known completely in advance. Analysis of information derived by processing the monitoring data may generate requirements for further information.

As the data sets retrieved from the database also tend to be quite voluminous, it is desirable to pre-process the collected data in a manner such that different information requirements may be fulfilled efficiently. This is the generation of Intermediate Information. Applications can derive different summaries from Intermediate Information without accessing the primitive database. This reduction in access to the primitive database increases the efficiency of the information processing applications.

Analysis of packet delay measurements is generally carried out for a specified window size throughout a monitoring period. For example, network service providers may need to know the average delays for all measurements carried out within a given week on an hourly basis. Thus the window size for this analysis is one hour whereas the investigation period is one week. Analysis window periods are integer fractions of 24.

An Intermediate Information structure should, therefore, allow calculation of simple statistical measures such as mean and variance, as well as the frequency distribution of delays within the window. If the window size is increased, Intermediate Information from smaller windows may be used to generate Intermediate Information for the larger window. This involves combining the Intermediate Information for the smaller windows. If the analysis window lies
between times \( t_1 \) and \( t_2 \), and \( S \) represents the set of all the packets received within the window then:

\[
S = \{ r_i \mid t_1 \leq r_i[t_2] < t_2 \} 
\]  

[3.10]

Let \( I^U \) represent the set of delay values \( d_i \) associated with the packets in \( S \).

\[
I^U = \{ d_i \mid (d_i = r_i[t_2] - r_i[t_1]) \land (r_i \in S) \} 
\]  

[3.11]

This set of delay values associated with received data packets are processed to generate appropriate Intermediate Information. Re-writing equation 3.2

\[
I = P(I^U) 
\]  

[3.12]

An information sub-system employing Intermediate Information to provide packet delay information receives a query specification \( Q \), for an investigation period between \( t_1 \) and \( t_2 \) and an analysis window size \( \text{win} \), from a client system. This query specification may be represented as follows:

\[
Q = (t_1, t_2, \text{tid}, \text{win}) 
\]  

[3.13]

The information sub-system decomposes \( Q \) into \( n \) component queries, \( q \), where:

\[
n = \frac{Q[t_2] - Q[t_1]}{\text{Q[win]}} 
\]  

[3.14]

and:

\[
q = (t'_i, \text{tid}, Q[\text{win}]) 
\]  

[3.15]

Thus each component query represents a query specification for a single analysis window within the query period. The decomposition operation, \( \delta \), may be represented as

\[
\delta(Q) = \{ q_i \mid (1 \leq i \leq n) \land (q_i[t*] = Q[t_i] + (i-1) \times Q[\text{win}]) \} 
\]  

[3.16]

The information subsystem provides an Intermediate Information object for each component query. Thus each query, \( Q \), posted to the information sub-system
results in a set of Intermediate Information objects. The number of Intermediate Information objects generated is equal to the number of analysis windows.

The finest granularity for analysis is considered here to be 1 hour. If Intermediate Information at a specific level of granularity is not found amongst that cached, the component query is further decomposed to basic component queries which have a window size of 1 hour each. If \( q' \) represents a query specification with a window size of 1 hour and \( I \) represents the Intermediate Information corresponding to \( q' \),

\[
q' = (i, tid, l) \quad \text{[3.17]}
\]

\[
\delta(q) = \{q_i', (1 \leq i \leq Q[\text{win}]) \land (q_i'[t_i] = q_i[t_i^*] + i - 1)\} \quad \text{[3.18]}
\]

Intermediate Information at the finest granularity i.e. 1 hour, is generated in response to each basic component query. The required Intermediate Information is constructed by suitably combining these finest granularity Intermediate Information elements.

\[
I = C(I_i') \quad \text{[3.19]}
\]

where \( 1 \leq i \leq Q[\text{win}] \)

This represents combination of Intermediate Information at the finest granularity.

### 3.5 Intermediate Information Structures for Packet Delay Information

This section describes two data structures that can be used as Intermediate Information structures for packet delay information. The first structure uses delay values associated with test packets, transmitted within an analysis window sorted in increasing order and is known as **Sorted Delays**. The second structure uses the frequency distribution of packet delay values and hence is termed as **Delay Distribution**.

The following paragraphs briefly describe these Intermediate Information structures and list the information elements that can be derived from them. Moreover, the characteristics of these structures that can assist in reusability and recyclability of different Intermediate Information elements are also highlighted.
3.5.1 Sorted Delays as Intermediate Information

This Intermediate Information structure uses packet delays for a specified window size, sorted in increasing order of delay. Intermediate Information elements that are instantiations of this structure allow calculation of following statistical measures regarding the delay experienced by test packets,

- mean
- variance
- the frequency distribution of delays within the window
- minimum and maximum delay values
- number of test packets received with in the specific analysis window
- percentiles

If the window size is increased, Intermediate Information from smaller non-overlapping windows may be used to generate Intermediate Information for the larger window. As the granular Intermediate Information elements are sorted in increasing numerical order, the merge-sort algorithm can be used to efficiently combine these elements to generate the required Intermediate Information elements.

3.5.2 Delay Distributions as Intermediate Information

The frequency distributions of delay experienced by test packets received by the receiving test station during a specified analysis window can also be used as Intermediate Information.

Frequency distributions are derived by generating the tabular grouping of primitive data into categories and reporting the number of observations in each category. Frequency distributions put data in order so that a visual analysis of the measurements can be performed. Frequency distributions also provide a convenient structure for simple computations.

A frequency distribution can be used to generate cumulative frequency distribution. This provides additional information and insight about the corresponding dataset. It contains the same number of classes as the frequency distribution, however the cumulative distribution shows the frequency of items with values less than or equal to the upper limit of the class.

The frequency distribution along with the cumulative distribution can be used to derive the following information,
• mean
• variance
• number of test packets received within the specific analysis window
• percentiles

Delay Distributions cannot provide the accurate minimum and maximum delay values within the specific analysis window. However, the class representative of the first non-zero class can approximately be used as the minimum delay value. Similarly, the class representative of the last non-zero class can approximately be used as the maximum delay value. In order to provide the required information with reasonable accuracy, suitably small classes or bins need to be used.

In the simplest case, these Delay Distributions are generated by choosing a maximum value of delay that a test packet is expected to experience. The bin size is a function of the maximum delay value and the number of bins required in the distribution.

In an analysis window, there may exist test packets which experience delays greater than the maximum delay value specified. In this case, the Delay Distribution is generated for the delay values lower than the specified maximum delay value. For delay values greater than the specified maximum, the highest value of delay and the number of test packets that experience delay greater than the specified maximum value are maintained.

Delay Distributions generated for smaller non-overlapping analysis windows can be added to provide the Delay Distribution of a larger analysis window, if the Delay Distributions of the smaller analysis window contain the equal number of bins and bin sizes.

3.6 Comparison of Information Systems Using Sorted Delays and Delay Distributions

The performance of an Intermediate Information system using Sorted Delays is compared with an Intermediate Information system using Delay Distributions. The Intermediate Information system using Sorted Delays as Intermediate Information is termed as the Sorted Delays System and the Intermediate Information system employing Delay Distributions as Intermediate Information is termed as the Delay Distribution System.
Information retrieved from these information systems consisted of the average delay of successive 5% fastest packets for all window sizes which are integer fractions of 24. For the specific implementation of the Delay Distribution System, it is assumed that the maximum possible delay a test packet experiences is 100 milli-seconds. For all delay values greater than 100 milli-seconds, the maximum delay value is stored as well as the number of test packets which experience delay greater than 100 milli-seconds.

For these experiments, the Delay Distribution System provides the required information by generating and processing Delay Distributions with the following bin sizes:
- 200 micro-seconds, i.e. 500 bins
- 100 micro-seconds, i.e. 1000 bins
- 50 micro-seconds, i.e. 2000 bins
- 25 micro-seconds, i.e. 4000 bins
- 20 micro-seconds, i.e. 5000 bins

Figure 3-4 displays the performance of the Sorted Delays System with respect to the performance of the Delay Distribution System when all Intermediate Information elements at the finest granularity are available to these system.

![Figure 3-4](image-url)
It can be seen that the Delay Distribution System provides a faster response than the Sorted Delays System in answering the queries posted to it. This is true for all bin sizes except for the bin sizes of 25 and 20 microseconds. In these cases the Sorted Delays System provides faster response for analysis windows of 2 and 3 hours. This may be due to the relatively less amount of data that the Sorted Delays System needs to access and process to generate the Sorted Delays for these analysis windows.

Figure 3-5 shows the performance of the Sorted Delays System compared to the performance of the Delays Distribution System when no Intermediate Information at any level of granularity is available to both these systems. The Delay Distribution System provides faster response as compared to the Sorted Delays System for all bin sizes and analysis windows.

Figures 3-6, 3-7 and 3-8 show the comparison of the performance of these systems when
- Sorted Delays at finest granularity are available to the Sorted Delays System but no Delay Distributions are available to the Delay Distribution System (Figure 3-6)
- Sorted Delays at the required level of granularity are available to the Sorted Delays System but only Delay Distributions at the finest granularity are available to the Delays Distribution System (Figure 3-7)
- Intermediate Information elements at the required levels of granularity are available to the Sorted Delays System as well as the Delay Distribution System (Figure 3-8)

It can be seen from figure 3-6 that if Sorted Delays are available at the finest granularity, the Sorted Delays System can provide a faster response as compared to Delay Distribution System for some analysis windows.

On the other hand, figure 3-7 shows that, for Sorted Delays pre-processed at the required granularity and Delay Distributions pre-processed at the finest granularity, the Delay Distribution System can provide a faster response than the Sorted Delays System. This is true if Delay Distributions are generated with bin sizes greater than 25 micro-seconds. For Delay Distributions generated with bin sizes of 20 and 25 micro-seconds, the Sorted Delays System provides slower response when compared with the Delay Distribution System if the required information is derived for the analysis window size of 24 hours.

Figure 3-8 shows the performance of Sorted Delays and Delay Distribution Systems when Intermediate Information elements at the required levels of granularity have been pre-processed. As the size of the Delay Distributions is dependant on the bin size and not on the granularity required (contrary to Sorted Delays), the Delay Distribution System provides uniform response if the specific Delay Distributions have been pre-processed at the required granularity. On the
contrary, the response of the Sorted Delays System varies with the analysis window for which information is required. This is because the size of a Sorted Delays element varies with the size of the analysis window.

Figure 3-7: Performance Comparison (Required Sorted Delays available, Delay Distributions at finest granularity available)

It can be seen that if the Intermediate Information elements at the required level of granularity are available to the Delay Distribution System, it provides a faster response as compared to the Sorted Delays System for all Delay Distributions with
bin sizes greater than 25 micro-seconds. The Sorted Delays System, however, provides a faster response when compared to the Delay Distribution System for Delay Distributions with bins sizes of 20 and 25 micro-seconds if the required information is derived for analysis windows of less than 4 hours.

Systems using Delay Distributions provide relatively faster response to queries as compared to the systems using Sorted Delays. However, the information derived from Delay Distributions is relatively inaccurate when compared to the information derived from Sorted Delays. Within a given interval, the frequencies may not generally be equally distributed across the interval. An assumption of equal spread of frequencies within an interval may therefore not be entirely correct and does lead to errors in computations. These errors are significant for wider intervals and increase with the width of the intervals.

For the dataset used in these experiments, figure 3-9 provides an indication of the accuracy of information derived from Delay Distributions of different bin sizes. It can be seen that as the number of bins is increased (bin size is reduced), the information derived from Delay Distributions is more accurate.

![Figure 3-9: Accuracy of Information Derived From Delay Distributions](image)

3.7 An Architecture for Network Performance Information Sub-system Employing Intermediate Information

An information sub-system employing Intermediate Information to process network performance information, like any other information system, receives a query from a client object. In response to the query received, the information sub-
The application of intermediate information concepts to process packet delay information

System provides the requested intermediate information objects to the client object. The client object then derives the required information elements from the intermediate information objects. The information sub-system also generates exceptions in response to abnormal conditions, which may also be thrown to the client object (Figure 3-10).

The query, an object of the `AggregateDelayQuerySpecs` class, received by the information sub-system is decomposed into its component queries by an object of the `QueryDecomposer` class (Component Query Generator). Each component query is a query specification of a particular analysis window within the queried period.

An object of the `IntermediateInformationFiler` class is used to store and retrieve relevant intermediate information objects (`DelayIntermediateInformation`) on the disk. All component queries generated by the Component Query Generator are passed to the object of the `IntermediateInformationFiler` class. This object retrieves appropriate intermediate information objects from the disk. These intermediate information objects are returned to the client object by the information sub-system.

Figure 3-10: Object Oriented Intermediate Information Sub-system
If the object of the `IntermediateInformationFiler` is unable to locate any appropriate Intermediate Information object on the disk, the component query is passed to another object of the `QueryDecomposer` class (Granular Query Generator). This object generates objects of the `AggregateDelayQuerySpecs` class representing query specifications to retrieve Intermediate Information at the finest granularity. These granular query specifications are used by the object of the `IntermediateInformationFiler` class to retrieve the required granular Intermediate Information objects from the disk (if these have been pre-processed and cached).

All granular Intermediate Information elements are combined by an object of the `IntermediateInformationCombiner` class. This object returns an Intermediate Information object for a specific analysis window by appropriately combining the granular Intermediate Information objects that are passed to it. These Intermediate Information objects at the required granularity are cached for subsequent reuse by the object of the `IntermediateInformationFiler` class. The information sub-system returns these Intermediate Information objects to the client object.

For any granular Intermediate Information object that has not been pre-processed and cached, the granular query specification is passed to the object of the `IntermediateInformationGenerator` class. This object generates the granular Intermediate Information object for the granular query specification passed to it by accessing the primitive database and appropriately processing the extracted primitive data. The granular Intermediate Information objects generated by this object are cached by the object of the `IntermediateInformationFiler` class. These granular Intermediate Information objects are combined appropriately by an object of the `IntermediateInformationCombiner` class to generate Intermediate Information objects for the queried analysis windows. These Intermediate Information objects at the required granularity are cached for subsequent reuse by the object of the `IntermediateInformationFiler` class.

The information sub-system described here uses the query it receives from the client object to initialise its different components. Intermediate Information objects corresponding to the analysis windows in the queried period have to be *pulled* from the information sub-system by successively triggering an appropriate method in the information sub-system.
3.8 The Impact of Information Reusability on the Performance of Information System

The performance of an information system employing information reusability by using Sorted Delays is compared with a system that does not employ information reusability. The system that uses Sorted Delays is termed here as the Sorted Delays System and the system that uses only the primitive database to process the required information is known as the Primitive Information System.

The performance of the Sorted Delays System is compared with the performance of the Primitive Information System for the following states of the Intermediate Information Base (Figure 3-11)

- No Intermediate Information is available. The Sorted Delays System needs to access the primitive database, calculate the Intermediate Information elements at the finest granularity, combine these to produce Intermediate Information at the required granularity and then derive the required information.
- Intermediate Information elements at the finest granularity are available. To provide the required information, the Sorted Delays System combines these Intermediate Information elements to generate Intermediate Information at the required granularity and then derive the required information from it.
- Intermediate Information at the required granularity is available. The Sorted Delays System simply processes the appropriate Intermediate Information elements to generate the required information.

Information retrieved from these information systems consisted of the average delay of the successive 5% fastest packets for all window sizes which are integer fractions of 24.

The Sorted Delays System uses the same sorting algorithm to calculate the granular Intermediate Information elements as used by the Primitive Information System. However, even in the worst case (i.e. no Intermediate Information available), the Sorted Delays System is seen to provide a faster response to the queries. This may be because the number of data elements (i.e. delay values) to be sorted to generate the granular Intermediate Information elements is much smaller than the number of data elements used by the Primitive Information System to generate the required information. Moreover, once the Intermediate Information elements at the finest granularity have been generated, their characteristic (i.e. the data elements constituting granular Intermediate Information are sorted) can be exploited while combining them. Thus the merge-sort algorithm used to combine
these granular Intermediate Information elements significantly reduces the response time.

Consider an information system that derives the required information from appropriate Delay Distributions generated directly from the primitive database. This system is termed here as the Non-reusing Delay Distribution System. The performance of this system is compared with one that reuses granular Delay Distributions to construct the appropriate Delay Distributions and then derives the required information elements. This system is termed here as the Delay Distribution System. This comparison provides an indication of the overhead of deriving the finest granularity Delay Distributions as well as the impact of reusing these Delay Distributions to provide the required information.

Figure 3-11: Response Time Measurements: Primitive Information System Vs Sorted Delays System

It can be seen from the graphs in figure 3-12 that if no Delay Distributions are available, the Delay Distribution System has lower performance than that of the Non-reusing Delay Distribution System. However if the appropriate finest granularity Delay Distributions are available, the performance of the Delay Distribution System is higher (i.e. lower response times) as compared to that of the Non-reusing Delay Distribution System.
The Application of Intermediate Information Concepts to Process Packet Delay Information

Figure 3-12: Response Time Measurements: Non-reusing Delay Distribution System vs Delay Distribution System
It can also be seen that as the size of Delay Distributions is increased, the overhead of deriving the appropriate Delay Distributions from the finest granularity Delay Distributions also increases. However if the appropriate finest granularity Delay Distributions are available, it is economical to reuse them to generate the appropriate Delay Distributions and then derive the required information.

3.9 Related Research
A number of researchers have investigated reusability of information at different levels in information systems. This section provides a brief overview of some of the work conducted in this direction.

Levitt et al describe an interactive data analysis system [LevSY74]. This system groups discretely identifiable units of data collected for analysis (cases) into sets on the basis of similarity of their attribute classes. Attribute classes are conceptual division of cases and an attribute class is defined by its name and associated set of values. These may be empty, consist of only one datum or a large class of related data.

This system provides the user with two basic classes of capabilities. One consists of operations that are executed on sets, i.e. creation and deletion of sets, display of set data and associated data summary statistics (min, max, sum, average etc.). The second class consists of a variety of statistical methods that can be applied to data contained in sets, i.e. histograms, scattergrams, plots, regression etc.

The system also provides methods for recombining different sets and attributes to produce aggregate sets. Moreover subsets can also be created from a set by not including in a subset cases that do not satisfy appropriate conditions. Sets can also be created from more than one parent sets by performing appropriate union and intersection operations. During the set creation process, a set of basic summary statistics are computed for every attribute in the set that is being created.

Wiederhold et al describe the Time Oriented Database (TOD) used for managing and processing clinical information [WieFW75]. This database system allows the generation of Access files on request. These files provide rapid access to data according to various retrieval criteria. All data in these files are redundant relative to data files themselves. These files are typically generated overnight and are
identified by the date of their creation. This allows reporting of the date of validity in reports which use them.

Different types of Access files can be used. **Index** files contain all values of the specified elements in ascending order with the symbolic keys to the records. **Range** files contain the lowest and highest values of specified attributes of patients' visit information along with the key to the patients' personal information. **Transposed** files are organised by elements. A transposed file contains one record per transposed attribute type from the main file. Each record thus has a number of data values which is equal to the number of records in the main file. If all elements in a file are transposed then the transposed file will be equal in size to the main file.

Transposition aids in selection or evaluation of data according to a dimension orthogonal to the data entry or update dimension. It allows for discovery of trends or functional dependencies among attributes otherwise not obvious [Wie83]. However for very large data files, transposed files may generate very long records. Thus generation of transposed files and processing of transposed records may pose significant challenges.

Brooks et al suggest using partitioned databases for statistical data analysis [BroBP81]. They suggest that generally while analysing statistical information, a preliminary analysis run indicates that a more refined coding scheme is needed or deficiencies in the data collection are revealed by the coding process. The user then performs the data analysis calculations repeatedly over time using new coding schemes, new variables and new selections of groups of measured data.

Brooks et al also suggest that while the variety of possible statistical analysis are extremely large, nearly all of the commonly used analysis begin with a common set of calculations, for e.g. sums. These calculations form the basis for further calculations in different analysis. Thus a typical analysis process begins with formation of suitable partitions or classes of the database, calculation of summaries and performing the necessary calculations on these quantities to generate the required information. For example, to perform an analysis of variance on a variable $x$, $\sum(x)$ and $\sum(x^2)$ are calculated for each of the partitions or classes being processed. These sums may then be used to generate the required information. If subsequent analysis requires collapsing these classes, the sums for
the classes being collapsed need only to be added together to provide the sums for the collapsed classes.

Other researchers have also demonstrated the reusability of summary data in statistical databases and data warehouses (for e.g., [CheMcN89], [RafRic93], [LenSho97], [MumQM97]). They show that additive summaries of a union of disjoint category instances can be computed directly from the known summaries of these category instances. Some non-additive summaries, for e.g. means and variances, can be decomposed to their additive primitives thereby facilitating reuse.

3.10 Conclusions

Reusability of information system components in the design and development of new systems ensures the rapid development of inexpensive and high quality systems. Reusability of processed information and data in deriving more information, on the other hand, can have a significant impact on the performance of an information systems in providing requested information.

Research has been conducted in achieving re-usability of information in information systems and databases. Most of these approaches attempt to reuse the additive summary information of disjoint subsets of data to generate similar summaries for subsets created by the union original subsets. Non-additive summaries like average and variances are converted to their additive primitives which can be conveniently re-used.

These approaches, however, are restrictive. In certain environments (for e.g. research and development) it may be required to summarise the data collected by monitoring a process in a variety of different ways so as to investigate its different attributes. This may necessitate access to the primitive database for the calculation of the required summaries (some of which may not be re-usable).

A model for an information system has been presented here that exploits the concepts of information reusability and recyclability. Such information systems process primitive data to generate Intermediate Information. Different finer granularity Intermediate Information elements may be combined (re-cycled) to generate coarser granularity Intermediate Information. Moreover, different information elements may be efficiently derived from suitable Intermediate Information elements, for e.g. different averages, variances and percentiles.
Intermediate Information elements represented in different structures can provide the same information elements. However different Intermediate Information structures have different characteristics. For example, some may incorporate loss of information and therefore may not provide accurate information. Others may not be as rich and can only provide limited information regarding the process being monitored.

Application of Intermediate Information has been studied in the context of network performance monitoring. Two suitable Intermediate Information structures have been described which are used to provide appropriate information regarding packet delays. Sorted Delays use the delay values of a particular analysis window sorted in increasing order. Delay Distributions use the frequency distribution of delay values in a particular analysis window.

Sorted Delays provide accurate information as they do not incorporate any loss. Delay Distributions on the other hand provide varying degrees of accuracy which is dependant on the bin size used in generating these distributions. Systems employing Delay Distributions generally have higher performance as compared to the systems employing Sorted Delays. Systems using Delay Distributions with small bin sizes (i.e. higher number of bins) may have lower performance if compared to systems employing Sorted Delays for small analysis windows when either the required Intermediate Information is available to these systems or there exist suitable granular Intermediate Information elements to generate the required Intermediate Information elements.
Chapter 4
Dynamic Selection of Intermediate Information Structures

4.1 Introduction
It has been shown, in chapter 3, that different Intermediate Information structures can be used to provide the same information with different levels of accuracy. The performance of these information systems also vary with respect to the Intermediate Information structures employed. This chapter attempts to discuss the possibility of using more than one Intermediate Information structure in an information system. Means to dynamically select a suitable Intermediate Information structure that efficiently provides the required information with reasonable accuracy are also discussed.

The system response time requirements in a DSS (Decision Support System) environment are generally more relaxed than those for OLTP (On Line Transaction Processing) systems [Inm96], [Dea64]. The performance of data warehouses and information systems supporting a DSS environment is however an important issue as it can be related to the productivity of the analysts [Inm96].

In an OLTP environment, the measurement of response time is conducted from the moment of submission of a request to the moment of return of corresponding results. On the contrary, several options exist for measuring response times and performance of data warehouses [Inm96]. In a DSS environment there is no clear time for measuring the return of data. This is because a large amount of data are often returned as a result of a query. Results of the queries may also be returned in parts.

One possibility is to measure the time between the submission of requests and the return of the first set of data. It is also possible to measure the time between the submission of request to the return of last set of data.

Another related issue is determining the location in a system where these measurements are conducted. Response time can be measured at the end user's terminals as well as on the data warehouse servers. Issues related to the measurement at either of these locations have been mentioned by Inmon [Inm96].
Performance information for data communications networks is generally derived for specific analysis windows within the queried period. The query issued by the client is therefore decomposed into component queries. Network performance information for each component query is derived and returned to the client. It may therefore be useful to measure/calculate the time required by the information server to process each component query. These estimates would allow optimisation of each individual component query so that a suitable Intermediate Information structure can be used to provide the required information for that component query.

Researchers investigating human computer interaction stress that overall productivity depends not only on the speed of the system but also on the rate of human error and ease of recovery from these errors [Shn92]. Lengthy response times are generally detrimental to productivity, increase error rates and decrease satisfaction. More rapid interactions are generally preferred and can increase productivity but may also increase errors. The high cost of providing rapid response times and the loss from increased errors must be evaluated in choosing an optimum pace. It is also suggested that modest variations in response times (± 50% of the mean) appear to be tolerable and have little effect on human performance.

It may be useful to slow down unexpectedly fast responses to avoid surprising the user [Shn92]. However this proposal is controversial and would affect only a small fraction of user interactions. More serious efforts are generally made to avoid extremely slow responses. If they occur, the user should be given information to indicate the progress towards the goal.

This chapter proceeds with explaining some research into improving information system performance by varying different aspects of the databases or data structures employed by them. A brief introduction to query optimisation is also provided. Different approaches to dynamic selection of Intermediate Information structures for network performance information are then described. Various aspects of a prototype information system that dynamically selects appropriate Intermediate Information structures are explained. This prototype constructs the Delay Distributions from the finest granularity Sorted Delays, if the required Delay Distributions are not available. This chapter is concluded after discussing the possibility of reusing coarser granularity Delay Distributions to generate Delay Distributions with granularity coarser than the ones being reused.
4.2 Related Research

Performance of databases and information systems can be enhanced significantly by employing a variety of techniques, for e.g., indices, partitions, clusters, replication and extraction (distributed databases) and segregation of organisational and departmental DSS activities. Additionally research has also been conducted in self organising data structures. These data structures organise themselves on the basis of access operations. This allows retrieving records with a higher probability of access more efficiently.

Research has also been conducted on database reorganisation and on-line reorganisation. This includes a wide range of activities from varying the database file structures to database maintenance in order to achieve desired performance.

Moreover, once a database has been constructed and populated, there may exist several ways to answer a query posted to it. Some of these may be more efficient than others. The task of determining possible ways of solving a query, calculating the cost of all possible query plans and the selection of an optimal plan has been extensively investigated under query optimisation.

This section attempts to briefly touch upon the progress made in these areas.

4.2.1 Self Organising Data Structures

Research in this area is concentrated towards organising lists and trees in response to operations conducted to access different records.

It is possible to keep a count of how often a record in a list has been accessed. The records in the list can then be re-allocated on the basis of these counts. Knuth suggests that such a procedure would often lead to substantial saving in access operations [Knu73]. However it may either not be appropriate or feasible to allocate extra memory space to the count fields.

It may be possible to re-organise lists in the order of the frequency of access of different records without using auxiliary count fields. Whenever a record has been successfully located, it is moved to the beginning of the table. The idea behind this technique is that frequently used records will tend to be located near the beginning of the table while records that are infrequently accessed
will drift towards the end. While no stable ordering is achieved, near optimal ordering occurs with high probability [Riv76].

Rivest proposes that a transposition rule is better than the move to front heuristic described above [Riv76]. This heuristic exchanges the desired record with the immediately preceding record. If the desired record heads the list, nothing is done. It is also suggested that the transposition heuristic is simple to implement. However Rivest suggests that if there is a high correlation between successive requests, the move to the front heuristic is sometimes more efficient [Riv76].

Bitner has studied the rate of convergence of different heuristics to their asymptotic costs. He proves that although the transposition rule has a lower asymptotic cost than the move to front rule, it decreases the cost more slowly [Bit79]. This makes the move to front rule a better choice if few requests are to be made to the list.

Bitner also proposes variations of a hybrid rule, where a rapidly converging heuristic (for e.g., move to front rule) is combined with one that has a lower asymptotic cost (for e.g. transposition rule). Thus initially a rapidly converging rule is used, asymptotically it behaves like a transposition rule and has a lower asymptotic cost. However determining when to switch rules is difficult and presents a major issue in this respect.

Several rules that use counters have also been described by Bitner. It is suggested that these rules provide better performance over permutation rules [Bit79]. It is also possible to combine permutation rules with the ones that use counters so that decisions to move the accessed records can be made on the basis of the contents of the counters. For example, wait c, move and clear rules associate a field, initially 0, with each record. When a record is accessed, the corresponding field is incremented. If it exceeds a specified maximum value, the record is moved using the corresponding permutation rule (transposition or move to front rules) and the counter for each record is cleared.

Rotations in tree structures have been used to define analogues to the transposition and move to front rules for lists. The move up one rule uses a rotation to move the requested key up one level and the move up root rule
successively applies rotation, promoting the requested record until it becomes the root. Literature includes several heuristics (for e.g. [Bit79], [SleTar85] and [CheOn93]) that attempt to dynamically organise and balance trees so that overall access can be accomplished efficiently.

4.2.2 Database Re-organisation
Prywes addresses the problems of automation of very large databases [Pry69]. In addition to other aspects, he concentrates on the need for an easily changeable database organisation based on automatically monitored operational parameters in order to retain efficiency.

Statistics are collected during system operation to allow evaluation of the operation of each component of the database as well as the total system. These statistics form the feedback for continuous adaptiveness of the system to changing demands, thereby improving efficiency in performing different operations.

Statistics collected include usage of specific files and records. Various parameters regarding the requests made by each user to the system are also monitored. Prywes suggests that these monitored parameters exhibit tremendous variation of demands on the information system.

This system employs a changeable file organisation to optimise the cost of the database operations. Cost is determined by multiplying the frequency of use of a function by the sum of the cost associated with storage and the cost associated with the processing time in accessing the required information.

The system utilises a variety of mass storage devices and can employ different storage and retrieval techniques. This allows changing the cost of a function over a wide range to adapt it to the changing rate of usage. System managers, after appropriate evaluation of various operational, economical and priority factors, change the pre-setting of programs that organise information.

Stocker & Dearnley describe a system that determines the most suitable structure for a particular set of data and re-structures it when desirable [StoDea73]. Their system also determines an appropriate access strategy for each search. They assume that much of the data entered into the system will be used infrequently or even not at all. Some of it will be used intensively at
certain times separated by long inactive periods, whereas some may be in common use.

This system aims at minimising the total cost of data management. This is accomplished by analysing the requests made to the database as well as the actual contents of the data files. Moreover the system attempts to determine an estimate of the cost of an access before it is performed.

The database system described is based on a concept of folios of data files. Initially the original data file is the sole constituent of the folio. A particular pattern of later accesses may result in the formation of different files in the folio derived from the original data file (also known as versions of the folio). The full information content of a folio is preserved in the original data file.

A folio is, therefore a set of files, often of different completeness, which may consist of overlapping versions. The versions are created by the system in response to different data access requirements or different cost considerations.

When data is requested, the system determines the logical correctness and feasibility of the request. It attempts to determine the cheapest access strategy on the basis of existing files in the folio. It also attempts to determine the cost of creating new files in one or more folios. The system presents the optimum choices to the user. Results are generated by employing a specific acceptable strategy.

When the system is idle, it reviews the usage made of the database. This allows creation of new files or reconstruction of old files to make the predicted usage of the database more economic for a group of users rather than individual users.

Dearnley describes a model of the self organising database explained above [Dea74]. He suggests that the behaviour of the model proves the viability of a system that can

- choose suitable file structures
- construct access strategies
- attempt to minimise the cost of access
- change the database according to database usage
Dynamic Selection of Intermediate Information Structures

Detailed investigation of different scenarios by Dearnley with the help of the above mentioned model suggest possible cost savings for different database operations [Dea74b].

Hammer discusses the design principles of an automatic system that has the ability to choose the physical design for a database and to adapt this design to changing requirements [Ham77]. He suggests that a particular design of a database enables the efficient execution of certain retrievals but not others and might require extensive maintenance for certain updating activities. The objective of the physical design process is the selection of a physical representation that provides a good performance in the context of the particular mix of retrievals and updates that the database is expected to be subjected.

The system described by Hammer consists of
- a module that collects global statistics on the overall usage patterns of the database
- a predictor that projects observed usage statistics into the future
- a design evaluator that computes a figure of merit for any proposed design
- a heuristic proposer that synthesises a small set of candidate designs for detailed consideration.

Hammer demonstrates the application of the system mentioned above in the selection of secondary indices for an inverted file DBMS (Data Base Management System).

Recently, in order to reduce system downtime and increase system performance and availability, research into on-line re-organisation has been conducted. Omiecinski summarises his research on concurrent file re-organisation in [Omi96]. He briefly explains concurrent on-line conversion of B+ tree files to linear hash files and vice versa. This conversion is motivated by changes in user access patterns.

An indexed file structure is chosen originally as it can efficiently handle range queries. If after some time, the predominant type of query becomes exact match, performance can be improved by employing a linear hash [Omi96].

89
4.2.3 Query Optimisation

Queries to database systems are usually presented as requests for information in a high level language that specifies various properties of the desired information. The first step in query optimisation is **rewriting** the query. This involves translating the commands of the query language into operators from query algebra. Identities known to produce improvements in the execution of the query define a space of equivalent queries. This space is the set of all queries (in algebraic form) that can be obtained from the original query by the application of the identities [Leu94].

The second step involves the **physical level optimisation** and is also known as **plan generation, algorithm selection** or **access path selection**. A plan generator translates the algebraic form of the query into a program that access the stored data and derives the desired information. In addition to selecting suitable algorithms for implementing various operators in the re-written query, the plan generator also attempts to exploit any special data structures, such as indices, to improve the performance. The plan generator searches a space of equivalent plans, which it obtains by choosing multiple algorithms and data structures for the operators in the query. It attempts to make an estimate of the cost of evaluating the query by each alternative and then selects the plan with the lowest cost [Leu94].

Kirkwood divides query optimisers into the following types [Kir93]

- **Positional** optimisers use the position of the selection criteria to choose the most optimum execution plan.

- **Syntactical** optimisers use the syntax of the selection criteria to choose the optimum execution plan. Each operator (for e.g., =, >, between etc.) is given a fixed percentage of records that will be retrieved, with equality being the most selective. The next most selective operator is the closed limit, greater than or less than being the next most selective and so on.

- **Statistical** optimisers use the statistics of the record distribution to choose the most optimum execution plan. It is suggested that this is the best optimisation approach as it uses the actual distribution of records. However for it to be effectively implemented, it is necessary that these statistics are dynamically updated.

Query optimisation is a heavily researched subject area. Significant amount of work is being conducted on optimisation techniques for object oriented
databases and information systems maintaining and processing non-standard data. Recently attempts have also been made towards applying principles of Artificial Intelligence to the query optimisation problem.

4.3 Approaches to Dynamic Selection of Intermediate Information Structures

The packet delay information from the network performance data can be pre-processed as Sorted Delays or as Delay Distributions. Elements from both these Intermediate Information structures can provide different information elements, for e.g., averages, variances, percentiles etc. Moreover finer granularity Sorted Delays and Delay Distributions can be conveniently combined to generate coarser granularity Sorted Delays and Delay Distributions respectively.

However relatively less time is required to derive Delay Distributions at a specific level of granularity as compared to deriving Sorted Delays at that level of granularity. But the accuracy of the information obtained from Delay Distributions vary with the bin size and number of bins, whereas Sorted Delays can provide accurate results. It may be appropriate in certain scenarios to provide approximate results quickly with reasonable accuracy rather than to provide accurate results slowly [Dea64] [Har97].

From an initial report (or a set of reports) generated by querying an information system, the analyst may detect abnormal conditions and/or interesting events. The analyst may then decide to investigate these events and attempt to analyse, at an appropriate level of granularity, only the information related to these events.

This process may be accelerated by deriving approximate information for the initial reports with reasonable accuracy. As the analyst drills down by requesting finer granularity information regarding events of interest, only a subset of data (or finest granularity Intermediate Information) used to generate information for the initial query may need to be processed. Thus it may be possible to provide relatively accurate information with a reasonable system response time.

The following paragraphs introduce two basic approaches towards dynamic selection of Intermediate Information structures for packet delay information. The overall operation of both the systems is based on the assumptions described above, i.e. the initial query (if requesting information for a long period) is processed from appropriate Delay Distribution elements. These Delay Distributions provide
approximate information. As the analyst attempts to investigate interesting events by processing information regarding these events, the system may derive the required information from Delay Distribution with more bins and smaller bin sizes or from appropriate Sorted Delays.

However even for initial longer queries, it may be possible to provide accurate information for analysis windows which have been investigated earlier at that specific level of granularity. If the cost of accessing these pre-processed accurate Intermediate Information elements is lower than the cost of deriving the required Intermediate Information elements, the system may access the available Intermediate Information elements and provide accurate results.

As mentioned previously, self-organising data structures re-organise elements within a data structure to provide efficient access to frequently accessed data elements. Database re-organisation techniques monitor the database usage for a specified period of time and then re-organise the database structure to enhance the efficiency of frequent operations. On the contrary, the techniques introduced in the following paragraphs attempt to determine an Intermediate Information structure that should efficiently provide the required summaries. The decision to select a particular Intermediate Information structure is based on the number of available Intermediate Information elements of a specified structure that can be re-used and the amount of primitive data or granular Intermediate Information that may be required to derive the requested summaries.

4.3.1 Independent Management of Sorted Delays and Delay Distributions

This approach suggests management of two independent Intermediate Information sub-systems by a higher level management layer (Figure 4-1). This layer presents the user interface and receives the information requests from the user.

Depending upon the amount of primitive data to be processed to answer the query, an initial query execution plan can be generated. For the initial query execution plan, the accuracy of the final information is considered to be inversely proportional to the amount of primitive data expected to be processed to answer the query. For example, if the user requires information for more than one month of monitoring data, the system may select the initial query execution plan to derive that information from Delay Distributions of
2000 bins. If however the queried period is between 2 weeks and one month, Delay Distributions of 5000 bins may be derived to provide the required information. Similarly for queried periods below 2 weeks, Sorted Delays may be used to provide the required information.

Once an initial query execution plan has been generated, the original query is decomposed into component queries, one for each analysis window within the queried period. The system attempts to generate a refined query execution plan from the initial query execution plan for each component query. A refined query execution plan provides information which is at least as accurate as can be obtained from the initial query execution plan for the specific analysis window. Thus during the query refinement process the system attempts to determine if it can derive the required information with greater accuracy faster than it can derive the information at the level of accuracy as specified in the initial query execution plan.

For the specific approach being discussed, the system determines the number of available finest granularity Delay Distributions and Sorted Delays that can be used to construct the respective Intermediate Information element for a specific analysis window. From these, it calculates the approximate time required to derive the required information by using each of the above mentioned options (i.e. from Sorted Delays or from Delay Distributions).
These estimates for the processing time are derived with the help of different system performance parameters.

This system also attempts to determine if Sorted Delays and Delay Distributions at the required level of granularity have already been pre-processed and are available. If these Intermediate Information elements are available, an estimate is also made for the time required to access these Intermediate Information elements and derive the required information.

The refined query execution plan is generated from an option that provides the required information which is at least as accurate as specified in the initial query execution plan but can be derived at the lowest cost (i.e. response time). Thus if the cost of deriving accurate information is lower than the cost of deriving less accurate information as specified in the initial query execution plan, the initial query execution plan is modified to derive accurate summaries.

Although this approach suggests significant enhancements in performance (i.e. reduction in response time), large amounts of redundant information may need to be maintained. Moreover query execution plan refinement may also be a relatively computationally expensive process.

4.3.2 Deriving Delay Distributions from Sorted Delays

With this approach, the sub-system providing Delay Distributions derives finest granularity Delay Distributions from finest granularity Sorted Delays and adds these finest granularity Delay Distributions to provide Delay Distributions at the required levels of granularity. Only these coarse granularity Delay Distributions are cached (Figure 4-2).

Consider an information system that derives the required information elements from Sorted Delays and an information system that derives the required information elements from Delay Distributions. The performance of these systems are compared with the performance of an information system that uses Delay Distributions constructed from finest granularity Sorted Delays to provide the required information (Figures 4-3 and 4-4).

It can be seen that for analysis windows greater than 4 hours, the response time of a system constructing Delay Distributions from granular Sorted Delays is lower than that of a system using only Sorted Delays as Intermediate
Dynamic Selection of Intermediate Information Structures

Information. It can also be seen that if Delay Distributions are derived from pre-processed granular Sorted delays, the response times of the system for different levels of granularity are closer to those of a system using only Delay Distributions (when finest granularity Delay Distributions are available). Moreover, the larger the number of bins in the Delay Distributions, the closer are the response times of the two systems providing the required information from suitable Delay Distributions (Figure 4-3).

However the reverse is true for the worst case performance. If no finest granularity Intermediate Information elements are available, the response times of the system which constructs Delay Distributions from finest granularity Sorted Delays are closer to those of the system using only Sorted Delays (Figure 4-4).

This approach may present lower performance (i.e. higher response times) than the one mentioned in the previous sub-section. However it needs to maintain significantly less redundant information because the finest granularity Delay Distributions are not cached. Additionally, the query execution plan refinement operation may also be a relatively inexpensive process as only the required Sorted Delays elements at the finest granularity for a specific analysis window may need to be searched.
Another advantage of using this approach is that the response times for subsequent queries (requesting only subsets of the primitive data/Intermediate Information used to derive the results of the initial query) are automatically reduced.

Figure 4-3: Constructing Delay Distributions From Sorted Delays (Finest Granularity Sorted Delays Available)
Figure 4-4: Constructing Delay Distributions From Sorted Delays (No Finest Granularity Sorted Delays Available)

This is due to the fact that this system derives finest granularity Sorted Delays before using these to construct the required Delay Distributions. If the subsequent queries request derivation of information for analysis windows within the period of the initial query, these finest granularity Sorted Delays
are re-used to accelerate the construction of more accurate Delay Distributions or Sorted Delays at the required levels of granularity:

In the previous approach, if the system derives the required information from Delay Distribution, the Intermediate Information Base (cache) for Sorted Delays is not affected. Thus if subsequent queries need to derive the required information from Sorted Delays, and finest granularity Sorted Delays are not available, these have to be constructed from primitive data. Moreover the finest granularity Delay Distributions for certain bin sizes cannot be re-used to generate Delay Distributions at other bin sizes.

Using a common Intermediate Information structure to maintain finest granularity Intermediate Information can therefore provide an overall enhancement in system performance.

Figure 4-5 shows the comparison of the performance of the sub-system deriving Delay Distributions from Sorted Delays for different bin sizes. It can be seen that, for this approach, the difference in response time for deriving Delay Distributions of different sizes is not very significant. However while processing large amount of data (i.e. very long queried periods) this difference can accumulate to give reasonable improvements in system performance.

Figure 4-5: Comparison of System Performance for Different Distribution Sizes (Delay Distributions Constructed from Finest Granularity Sorted Delays)
4.4 A Prototype System Employing Dynamic Selection of Intermediate Information Structures

This section describes the architecture of an information sub-system that dynamically selects the Intermediate Information structures to process packet delay information. The system described here is based on the approach discussed in section 4.3.2. This approach suggests the storage of the finest granularity Intermediate Information elements only as Sorted Delays. These may then be used to provide the required Intermediate Information elements of a specific structure.

The following paragraphs describe the method used to calculate the cost of processing a query with a specific Intermediate Information structure. Later the architecture of the entire system is described in detail.

4.4.1 Query Cost Calculation

This system uses the processing time for deriving the required information as the cost of processing the query. An estimate of the cost of processing the query with a specific Intermediate Information structure can be provided by the model of information systems employing Intermediate Information (Section 3.3). The cost of processing a query using a specific Intermediate Information structure can be determined by the following expression.

\[ R = a T_r + (N - a) T_h + T_g + T_q \]  

where

- \( R \) is the system response time,
- \( a \) is the number of finest granularity Sorted Delays elements available for a specific analysis window,
- \( N \) is the analysis window size,
- \( T_r \) is the time to read one pre-processed finest granularity Sorted Delays element,
- \( T_h \) is the time required to derive a finest granularity Sorted Delays element from primitive data,
- \( T_g \) is the time required to combine finest granularity Sorted Delays to generate a coarser granularity Intermediate Information element,
- \( T_q \) is the time required to derive the required information elements from a specific Intermediate Information element.
The parameters described above can be determined empirically for different Intermediate Information structures. The values of these parameters also vary with the number of primitive data elements that are used to construct the finest granularity Intermediate Information elements.

Figure 4-6: System Performance for Computing Information from Sorted Delays (Measured vs. Modelled Response)
Figure 4-6 shows the performance of an information sub-system using only Sorted Delays as Intermediate Information. These graphs show the time required by the sub-system to construct coarser granularity Sorted Delays from finest granularity Sorted Delays for different numbers of available finest granularity Sorted Delays within the analysis window. The response times measured from a prototype are compared with those calculated from the model mentioned above. It can be seen that the system response time decreases if a greater number of pre-processed finest granularity Sorted Delays are available for the specific analysis window.

Similar results have been processed for Delay Distributions of 2000 and 5000 bins (Figures 4-7 and 4-8 respectively). These Delay Distributions are calculated from the finest granularity Sorted Delays. These results also show that the system response time is inversely proportional to the number of available finest granularity Sorted Delays for processing that component query.

The model described above can be used to ascertain the cost of processing a specific component query if appropriate system performance parameters are provided to it and when the required Intermediate Information elements need to be constructed from the finest granularity Sorted Delays.

If the required Intermediate Information elements have been pre-processed and cached in the Intermediate Information Base, the information sub-system may only need to read these elements and provide the requested result. There would however be a cost associated with it and needs to be taken into account in order to generate a suitable query execution plan.

As mentioned previously, the parameters of the model vary for different numbers of primitive data elements that are used to drive the finest granularity Sorted Delays. Thus if the rate at which the test packets are transmitted is changed, new values for the parameters of the model will have to be determined.

This problem may be solved by maintaining the system performance parameters for different transmission rates historically. Appropriate parameters can be retrieved to assess the cost of processing a query from this historical data. Alternatively, those parameters related to some specific sampling rates may be determined in advance and stored appropriately. If the primitive data
Dynamic Selection of Intermediate Information Structures

for answering a specific query have been collected at one of these sampling rates, appropriate performance parameters are retrieved and used to assess the cost of executing the query. For other transmission rates, approximate performance parameters may be interpolated or extrapolated appropriately.

Figure 4.7: System Performance for Computing Information from Delay Distributions of 2000 Bins (Measured vs. Modelled Response)
4.4.2 System Architecture

A prototype object oriented information sub-system to process packet delay information has been developed in Java. At the highest level, the information sub-system is an object of the DelayInformationManager class. The architecture of this class is shown in figure 4-9.
An object of the `DelayInformationManager` class can be instantiated by passing the following objects as parameters to its constructor:

- **String** representing the database directory
- Query specification as an object of the `AggregateDelayQuerySpecs` class
- **Strings** representing names of files containing the specifications for generating initial query execution plan and the system performance parameters.
- A list (as an object of the `Vector` class) of the number of bins for which Delay Distributions are required to be constructed.

![Diagram](image)

**Figure 4-9: DelayInformationManager Class**

On receiving these parameters, the constructor instantiates the following objects:

- **PlanGenerator** object of the `DataStructureSelector` class. This object generates the initial query execution plan.
- **QueryBreaker** object of the `QueryDecomposer` class. This object breaks the input query into its component queries, one for each analysis window.
- **QueryProcessor** object of the `DelayQueryProcessor` class. This object generates the required Intermediate Information elements of the specified Intermediate Information structures for each component query.
- **PlanOptimiser** object of the `QueryPlanOptimiser` class. This object, on the basis of the initial query execution plan and the pre-processed Intermediate Information elements available, generates a refined query...
Dynamic Selection of Intermediate Information Structures

At retrieval, the client object can retrieve successive Intermediate Information elements by triggering the `getNext()` method of the object. These elements are type casted as objects of the `DelaySummary` class. `DelaySummary` is the super-class for the `DelayAggregate` class (Sorted Delays) and the `DelayDistribution` class.

The `getNext()` method of the object of the `DelayInformationManager` class triggers the `getNext()` method of the `QueryBreaker` object. This method returns an object of the `AggregateDelayQuerySpecs` class representing a component query for the next analysis window in the queried period. The `getNewPlan()` method of the `PlanOptimiser` object is then triggered. The initial query execution plan and the component query specification for the next analysis window are passed to this method as parameters. This method returns an object of the `QueryPlan` class which represents the refined query execution plan. Appropriate Intermediate Information objects type casted as objects of the `DelaySummary` super-class are then retrieved by triggering the `getResults()` method of the `QueryProcessor` object. The refined query execution plan and the component query specification are passed to this method as parameters.

The following paragraphs explain the `QueryPlanOptimiser` and the `DelayQueryProcessor` classes:

**4.4.2.1 QueryPlanOptimiser Class**

Objects of this class generate a refined query execution plan from the initial query execution plan for every component query specification in the initial composite query specification. The architecture of this class is shown in figure 4-10.

An object of the `QueryPlanOptimiser` class is instantiated by passing the following objects to the constructor as parameters

- **String** representing the database directory
- **Vector** containing bin numbers for which Delay Distributions are to be constructed
Dynamic Selection of Intermediate Information Structures

• **String** representing the name of the file contain the system performance parameters

Upon instantiation, this object instantiates the **InformationFinder** object of the **InformationDetector** class. This object attempts to determine if an Intermediate Information object of a particular structure for a specific component query has been pre-processed. Trigger to the confirmPresence() method of this object returns an object of the **InformationConfirmation** class. Objects of the **AggregateDelayQuerySpecs** class (representing the component query) and the **QueryPlan** class (representing the query execution plan) are passed to this method as parameters. Objects of the **InformationConfirmation** class provide an indication of the presence of respective Intermediate Information elements on the disk.

![Figure 4-10: QueryPlanOptimiser Class](image)

The **PerformanceReader** object of the **StructurePerformanceReader** class is then instantiated. This object opens a file containing the system performance parameters. It reads this file and returns an object of the **DataStructurePerformance** class with every trigger to its getNext() method.
Objects of the \texttt{DataStructurePerformance} class contain system performance parameters for a specific Intermediate Information structure. A trigger to the \texttt{getResponseTime()} method of an object of \texttt{DataStructurePerformance} class returns an estimate of the system response time to process that specific Intermediate Information structure for a specific level of granularity. The time required by the system to access a pre-processed Intermediate Information element can be retrieved by triggering the \texttt{getMinTime()} method of the specific object of the \texttt{DataStructurePerformance}.

The \texttt{PlanRefinery} object of the \texttt{PlanRefineryEngine} class is constructed by passing a list of objects of the \texttt{DataStructurePerformance} class as a \texttt{Vector} object. This list contains one object of the \texttt{DataStructurePerformance} class for every Intermediate Information structure, system performance parameters of which are read by the \texttt{PerformanceReader} object.

An object of the \texttt{QueryPlanOptimiser} class attempts to determine the existence of pre-processed Intermediate Information elements of the required granularity for all bin sizes equal to or smaller than those specified in the initial query execution plan. This is accomplished by generating appropriate query execution plans for each of these bin sizes and passing these successively, with the component query specification, as parameters to the \texttt{confirmPresence()} method of the \texttt{InformationFinder} object. Each object of the \texttt{InformationConfirmation} class returned by the \texttt{InformationFinder} object is passed as a parameter to the \texttt{addOption()} method of the \texttt{PlanRefinery} object. If the existence of the specific Intermediate Information element is confirmed, the response time for executing that query plan is determined with the help of the corresponding object of the \texttt{DataStructurePerformance} class. The query execution plan and the corresponding system response time in executing this plan are appended to a table of available query execution plans.

The object of the \texttt{QueryPlanOptimiser} class uses a \texttt{QueryBreaker} object of the \texttt{QueryDecomposer} class to decompose the component queries into query specifications with analysis windows of one hour each. These query specifications are combined into a \texttt{Vector} object. The existence of the finest granularity Sorted Delays corresponding to these component queries
of one hour windows is determined with the help of the InformationFinder object. The InformationFinder object returns a list of objects of the InformationConfirmation class as a Vector object.

This list can be used to determine the cost of constructing Intermediate Information elements of a particular structure with a specified granularity from the finest granularity Sorted Delays and/or primitive data. This is accomplished by passing this list to an overloaded addOption() method of the PlanRefinery object. This method determines the cost of processing a query by using a particular Intermediate Information structure if elements of this structure have not been pre-processed for the specified level of granularity. The query execution plan and the system response time for executing that plan are appended to the table of available query execution plans.

The object of the QueryPlanOptimiser class also attempts to determine the existence of Sorted Delays for the specific analysis window. If the Sorted Delays element for the specific analysis windows exists, the cost of retrieving it is appended to the table of available query execution plans. If the Sorted Delays element for the specific analysis window does not exist, the cost of deriving it from finest granularity Sorted Delays and/or primitive data is calculated as mentioned above. This cost is appended to the table of available query execution plans.

The getBestOption() method of the PlanRefinery object scans the table of the available query execution plans and returns the query execution plan with the lowest system response time for responding to the specific component query. The table of the available query execution plans and the corresponding system response times is then reset for determining the query execution plan for the next component query.

This process is summarised in the flow diagram shown in figure 4-11.

4.4.2.2 DelayQueryProcessor Class

An object of the DelayQueryProcessor class is instantiated by passing the database directory as a parameter to the constructor (Figure 4-12). This object uses objects of the DelayAggregateManager4 class (AggregateGenerator object) and the Delays2DistributionManager3
Dynamic Selection of Intermediate Information Structures

class (DistributionGenerator object). The AggregateGenerator object provides Sorted Delays and the DistributionGenerator object provides the Delay Distributions as Intermediate Information elements.

Figure 4-11: Refining the Query Execution Plan

An Intermediate Information object type casted as an object of the DelaySummary super-class can be retrieved by triggering the getResults() method of an object of the DelayQueryProcessor class. This method requires the query execution plan and the component query specification to be passed as parameters. Depending upon the query execution plan (i.e. to
derive Sorted Delays or Delay Distributions), either the **Aggregate Generator** object or the **DistributionGenerator** object is instantiated for a specific component query. Appropriate Intermediate Information objects (i.e. either of the DelayAggregate or the OpFreqDistStructure class) are generated by triggering the `getNext()` method of the Aggregate Generator object or DistributionGenerator object respectively.

![Diagram](Image)

**Figure 4-12: DelayQueryProcessor Class**

### 4.5 Reusing & Recycling Coarse Granularity Delay Distributions

The dynamic Intermediate Information selection approach and its implementation discussed above attempts to achieve a compromise between system response time, information accuracy and redundancy. The system described above constructs the required Intermediate Information elements from the finest granularity Sorted Delays.

Intermediate Information elements of a specific structure can also be constructed by combining appropriate Intermediate Information elements of the same structure at various granularities finer than the required level of granularity. For example, Delay Distributions of two consecutive 6 hour windows can be added to construct a Delay Distribution for an analysis window of 12 hours. Similarly, Delay Distributions for three consecutive 4 hour windows can also be added to provide the same Delay Distribution for the 12 hour analysis window. Intermediate
Information corresponding to analysis windows of un-equal sizes can also be combined. For e.g., one Delay Distribution for an 8 hour window and two for 2 hour windows (non-overlapping, adjacent windows) can also be added to give a Delay Distribution of a 12 hour window.

Under certain circumstances this approach may be more efficient than constructing the required Delay Distributions from finest granularity Sorted Delays. However determining a combination of the different finer granularity Delay Distributions and the finest granularity Sorted Delays that may allow efficient construction of the required Delay Distributions may be a complex problem. This is because a specific Delay Distribution may be constructed from a large number of different combinations of finer granularity Delay Distributions and the finest granularity Sorted Delays.

A simpler (but sub-optimal) approach may allow construction of the required granularity Delay Distributions by using finer granularity Delay Distributions of equal analysis window sizes. Thus the query plan optimiser would attempt to determine a level of granularity, from the available finer granularity Delay Distributions, at which the Delay Distributions can be efficiently combined to construct the required Delay Distributions.

Consider \( win \) to be the required analysis window size and \( win' \) an integer fraction of \( win \) (greater than 1). For a specific level of granularity, let \( T_{rd} \) be the time required to read a pre-processed Delay Distribution and \( T_{hd} \) be the time required to construct a Delay Distribution from the finest granularity Sorted Delays and/or primitive data. \( T_g \) represents the time required to add a specific number of finer granularity Delay Distributions to generate a coarser granularity Delay Distribution. \( T_q \) represents the time required to derive the required information elements from the processed Delay Distribution. The time required by the system, \( R \), to derive a Delay Distribution from a available finer granularity Delay Distributions is given as

\[
R = a T_{rd} + (N - a) T_{hd} + T_g + T_q
\]

where

\[
N = \frac{win}{win'}
\]
Figure 4-13 shows the response times calculated from the model in equation [4.2] to generate Delay Distributions of 2000 bins for different levels of granularity from finer granularity Delay Distributions. These response times have been calculated when only half of the required finer granularity Delay Distributions are available and the remaining are derived from the respective finest granularity Sorted Delays. Moreover it is assumed that all of the required finest granularity Sorted Delays are available. These response times are compared with those obtained from a system that only uses the finest granularity Sorted Delays to derive the required Delay Distributions.

![Figure 4-13: Derivation of Delay Distributions from Finer Granularity Delay Distributions](image)

It can be seen that the larger the analysis window size of the finer granularity Delay Distributions, the lower the time needed to generate the required Delay Distributions and then derive the required information. It can also be seen that for the scenario described above, it is almost as economical to re-use the Delay Distribution of analysis window size of 2 hours to derive Delay Distributions of 2000 bins as it is to re-use finest granularity Sorted Delays.

Figure 4-14 shows the response times derived from equation [4.2] for generating Delay Distributions of 5000 bins for different levels of granularity from finer granularity Delay Distributions. Only half of the required finer granularity Delay Distributions are available and the remaining finer granularity Delay Distributions are derived from respective finest granularity Sorted Delays.
Dynamic Selection of Intermediate Information Structures

Figure 4-14: Derivation of Delay Distributions from Finer Granularity Delay Distributions (Bins = 5000, %age of Finer Granularity Delay Distributions Available = 50, %age of Finest Granularity Sorted Delays Available = 100)

This graph shows a behaviour similar to that in figure 4-13. It can be seen here as well that system performance improves if finer granularity Delay Distributions for relatively larger window sizes (and hence smaller number of Delay Distribution elements) are used to construct a required Delay Distribution element. For the scenario described above, the required Delay Distributions of 5000 bins can only be efficiently derived from finer Delay Distributions of few wide analysis windows. Thus it is only economical to derive Delay Distributions of 5000 bins for analysis windows of 12 and 24 hours from finer Delay Distributions of 5000 bins for respective 6 and 12 hour windows if only half of these Delay Distribution elements are available along with the remaining Sorted Delays elements.

Moreover, as the number of available finer granularity Delay Distributions increases, the process of re-using Delay Distributions becomes more efficient (Figures 4-15 and 4-16). Figure 4-15 shows this behaviour for generating Delay Distributions of 2000 bins at different levels of granularity.

Figure 4-16 shows a similar behaviour of the information system for generating Delay Distributions of 5000 bins. However, generating Delay Distributions of 5000 bins from Delay Distributions of respective analysis windows of 2 hours is inefficient as compared to generating these from the finest granularity Sorted Delays.
It can be seen from these results that as the size of the Delay Distributions increase, the analysis window size of the finer granularity Delay Distributions that can be efficiently re-used also increase.

4.6 Conclusions
The structure and the organisation of data maintained by an information system generally provides optimum performance for only a certain subset of operations.
Moreover, specific characteristics of data and information also define their structure and organisation in an information system. Research has been conducted to develop information systems that are able to dynamically re-organise their various components in order to enhance their performance for varying requirements.

This includes re-organisation of different data structures (for e.g. lists and trees) so that the most frequently retrieved records can be accessed efficiently. Similarly database systems have also been developed that re-organise the information and data so as to reduce the overall cost of accessing these data and information.

The re-organisation activities mentioned above are generally conducted by monitoring and analysing the users' request to the information system. Some of these systems reconstruct the structure of the data to answer a specific set of queries if the cost of constructing new data structures is less than that of answering the query with the existing data structure. There may also exist situations where more than one version of the data files (with different organisation and structure of data) exist and the database may choose the version that allows the most efficient response to the query.

Information regarding a process is generally required as summaries derived from primitive data at different levels of granularity. Usually the information derivation process is iterative in nature, where the analysis of one report may generate further information requirements to investigate various interesting events. It may therefore be appropriate, in a number of circumstances, to generate the initial report with a reasonable level of accuracy in order to achieve lower system response time. Subsequent analysis operations may request information corresponding to data which is the subset of data needed to derive summaries for the initial report. It may be possible to derive this information with a higher level of accuracy and higher overall performance.

This chapter has concentrated on describing techniques that allow the dynamic selection of appropriate Intermediate Information structures to provide information regarding network packet delay. Two Intermediate Information structures have been investigated. Sorted Delays provide accurate information but require significantly long processing time in order to be constructed from finest granularity Sorted Delays and/or primitive data. Delay Distributions, on the other hand, require relatively less time to be constructed from finest granularity Delay
Distributions and/or primitive data. However their accuracy (as well as their processing time) vary with their size.

Intermediate Information elements are derived for specific analysis windows within the queried period. It is possible to devise a system that selects a suitable Intermediate Information structure for packet delay information regarding specific analysis windows on the basis of the original query as well as the number of granular Intermediate Information elements available.

The techniques introduced here are different from those usually employed by self-organising database systems, where the structure and organisation of data are varied in response to the variation in user requests over a period of time.

One possible solution is to manage two independent information sub-systems. One of these sub-systems can provide Sorted Delays as Intermediate Information whereas the other can provide Delay Distributions as Intermediate Information. On the basis of the original query, the system selects a query execution plan. This plan specifies the Intermediate Information structure for the worst case condition (i.e., none of the finest granularity Intermediate Information elements are available).

For every component query within the queried period, the system determines, on the basis of the number of available finest granularity Sorted Delays and Delay Distributions, whether to derive Sorted Delays or Delay Distributions. The size of Delay Distribution is kept either equal to or greater than that of the one specified in the initial query execution plan.

This approach can provide reasonably accurate information with significantly high performance. However this approach requires maintenance of large amounts of redundant information.

Alternatively it is possible to derive the finest granularity Delay Distributions from the finest granularity Sorted Delays which may be used to generate coarser granularity Delay Distributions. Only these coarser granularity Delay Distributions are cached for subsequent reuse.

Although for individual queries this approach may process the required information slower than the previous approach (i.e., management of separate Intermediate Information structures), the overall performance of drill down
analysis of the packet delay information may be enhanced. This is because the initial query constructs the finest granularity Intermediate Information with a structure (i.e. Sorted Delays) that is used for the construction of both Sorted Delays and Delay Distributions. Moreover significantly less amount of redundant information may need to be managed as finest granularity Delay Distributions are not cached.

It is also possible to use coarser granularity Delay Distributions to derive Delay Distributions at even coarser levels of granularity. This process is however more efficient than deriving the required Delay Distributions from finest granularity Sorted Delays if the component Delay Distributions used have been derived for larger component windows within the analysis window. Moreover the efficiency of this process also depends upon the number of component Delay Distributions available as well as the size of the Delay Distributions.
Chapter 5

Management of Packet Loss and Duplication Summaries

5.1 Introduction

The performance of data communication networks deteriorates considerably under load. This performance deterioration is reflected in delays experienced by data packets communicated over these networks. Under conditions of severe congestion, different network elements (i.e. routers and gateways) run out of resources (for e.g. buffers, etc.) and drop packets due to lack of resources required to manage these packets.

Intrusive network monitoring attempts to determine network characteristics by injecting test packets into the network. Monitoring systems then attempt to determine the behaviour of the network by monitoring these test packets as they are communicated through the network. Various events like congestion and network failure etc. can be determined by calculating the number of lost packets. Large number of losses or high loss rate may suggest congestion or network failure.

Connectionless packet switched networks may also generate duplicate packets which are discarded by the protocol at the destination node or by the application at the host. Duplication of packets may either be a characteristic of the routing algorithm or may occur due to delayed acknowledgement of packet reception at the destination host. In the later case the source host may time out and, assuming that the packet was lost, will re-transmit the packet again.

This chapter explains objects which can generate and manage packet counts data. Packet counts (macro data) are derived from the primitive data (micro data) collected by intrusively monitoring a network. Packet counts include the number of test packets transmitted over a specific route, number of test packets received by the receiving test station, number of test packets lost and the number of test packets duplicated during communication. These data are calculated for specific analysis windows during the analysis period.

Due to the additive nature of packet counts, packet counts during a specific period for a large analysis window size can simply be generated by adding packet
Management of Packet Loss and Duplication Summaries

Counts derived earlier for smaller window sizes for the same period, if these smaller windows do not overlap.

This chapter proceeds with explaining the characteristics of packet counts data. A model for storage and management of these summaries is then described. This chapter is concluded after explaining objects that have been developed to manage these summaries.

5.2 Characteristics of Packet Loss and Duplication Data

Intrusive network monitoring measures performance by transmitting test packets over the network under test. These test packets are transmitted by the transmitting test stations and received by the receiving test station. These may physically be the same computing entity (e.g. system developed by [PhiPR95]).

The transmitting and the receiving test stations maintain the log of the transmitted and received packets. These logs are integrated into the database of the monitored data for subsequent performance analysis. This primitive database maintains the record of every test packet transmitted and received as two separate tables.

As mentioned previously, for a particular test identifier, the transmit time of the test packets can be used as their identifier because it is different for every test packet. The packet identifier of a test packet represents the state of a counter that is incremented each time a test packet is transmitted over the network under test. This counter may reset itself as it overflows over long monitoring periods or when the test station is restarted.

The received and transmitted data tables are chronologically ordered. However it is possible that there exist two or more test packets in the received data tables that have the same transmit time but different receive times. This is due to the duplication of test packets during communication over the network under test.

Similarly there may also exist a condition such that there does not exist a received test packet in the received data table for a test packet that exists in the transmitted data table. This is due to the loss of test packets during communication over the network under test.

The calculation of the number of packets lost or duplicated is simply an operation of counting the occurrence of events defined above. These counting operations
generate statistical entities. Statistical entities are described by distinguishing between category attributes and summary attributes [RafBT96].

Each statistical entity consists of one or more summary attributes and a set of category attributes [RafBT96]. Summary attributes represent the result of an aggregation function on the micro data whose numerical values are called the summary data. Category attributes provide a qualitative description of the summary attributes. These are a part of the metadata, which enables the correct interpretation of both micro and macro data.

Shoshani highlights the following regarding the summary and category attributes [Sho91]

- A combination of the category attribute values is necessary for each of the values of each summary attribute. Category attributes, therefore, serve as composite keys for summary attributes and each summary attribute is functionally dependant on the category attributes. This relationship between the category and summary attributes is part of the semantics that needs to be modelled.

- There may exist significant redundancy in the values of the category attributes when they are represented in a relation. There may also exist situations where all possible combinations of the category attributes (i.e. full cross product) are valid. In such cases each value of a category attribute repeats as many times as the product of the cardinality of the remaining category attributes.

- The range of category attributes is usually small, from as little as two (e.g. sex) to a few hundred (e.g. days in a year). Summary attributes often have large ranges as they always represent numeric measures. Often category attributes are grouped together so as to have fewer categories (e.g. age groups rather than age). Category values are more descriptive in nature and therefore tend to be character data whereas summary values are numeric.

As mentioned earlier, summaries are required to be generated for

- test packets transmitted by the transmitting test station
- test packets received by the receiving test station
- test packets lost as they are communicated over the network links
- test packets duplicated as they are communicated over the network links

These summaries are required for specific analysis windows over the analysis period. Category attributes, in this particular case, include
• the analysis window size (in hours)
• test identifier (this include the test group, route being tested as well as the test packet size)
• start time of the analysis window

Packet count summaries are additive in nature. Additivity allows additive statistical functions to generate the summary of a set of tuples directly from the summaries of its partitioning subsets. Thus packet counts for larger analysis windows during the analysis period can be generated from the packet counts that have previously been calculated for smaller analysis windows during the same analysis period.

For example, consider the table shown in figure 5-1a. This table represents packet counts for a particular test. These summaries are calculated over a window size of three hours for a particular day. If packet counts are required for an analysis window size of six hours, these can be generated conveniently by adding together summaries generated for consecutive analysis windows of three hours (figure 5-1b).

<table>
<thead>
<tr>
<th>Time</th>
<th>Transmitted</th>
<th>Received</th>
<th>Lost</th>
<th>Duplicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00</td>
<td>3600</td>
<td>3591</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>03:00</td>
<td>3540</td>
<td>3523</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>06:00</td>
<td>3600</td>
<td>3590</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>09:00</td>
<td>3600</td>
<td>3599</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12:00</td>
<td>3100</td>
<td>3100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15:00</td>
<td>3600</td>
<td>3600</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18:00</td>
<td>2500</td>
<td>2501</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>21:00</td>
<td>3600</td>
<td>3600</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5-1a : Additivity of Packet Count Summaries

5.3 Modelling Packet Counts Data
Packet counts data represent statistical summaries generated over a specific time period for a particular window size. Management of statistical summaries represents new challenges. The main issues in this work range from the definition of appropriate data models to the physical organisation of data for efficient storage and access.
Data models use abstractions to hide implementation details and concentrate on the general common properties of data objects. Thus data models are strongly related to the types of entities found and the type of data collected in a particular application [Raf91].

A data model defines [Sat91]
• a notation for describing data
• a set of operations to manipulate that data

A number of data models for statistical data management have been proposed in the literature. Brief descriptions of some of these models have been presented by [Raf91] and [Sat91]. These models provide powerful abstractions for representation and manipulation of complex statistical objects and summaries.

Significant differences exist between the statistical summaries and the primitive data from which these summaries are derived. Models for and organisation of statistical summaries may, however, be strongly affected by
• the organisation of primitive data
• the characteristics of the processes that generate the primitive data and the summaries
• potential applications that may access these summaries

The intrusive network monitoring process can be modelled as sets of experiments or tests. Each experiment or test attempts to intrusively monitor a specific route of the network by injecting test packets of particular size. Each test is identified by a unique test identifier, which provides information regarding
• the test group or the test set to which that particular test belongs
• the route that is being tested
• test packet size used for the experiment

The primitive data collected for each test is stored in separate files. Primitive data for each test are analysed separately to generate the required summaries. Operations at the highest level of analysis may include
• detection of various events (excessive delays, excessive losses etc.) for each test
• correlation between summaries generated for different tests

At the lower level, the packet counts data may conveniently be handled as statistical tables. One statistical table (referred to as the Packet Counts Table) is generated for each test that is conducted by the monitoring system and the window size required for subsequent analysis.

\( e \) is a counts entity representing a tuple in a Packet Counts Table

\[ e = (t_s, n_t, n_r, n_l, n_d) \] \( 5.1 \)

\( t_s \) represents the start time of a specific analysis window, \( t_a = t_{a-1} + \text{win} \)

\( \text{win} \) represents the analysis window size, which is a factor of 24

\( n_t \) represents the number of transmitted packets within a particular analysis window

\( n_r \) represents the number of received packets within a particular analysis window

\( n_l \) represents the number of packets lost within a particular analysis window

\( n_d \) represents the number of packets duplicated within a particular analysis window

As these tables are generated for specific tests and analysis window sizes, the test identifier and the window size may be regarded as the implicit categories for these tables [RafBT96]. These categories are not used for classification within a
statistical table, but for classification of statistical objects (Packet Counts Objects) that have been extracted from different Packet Counts Tables.

Packet counts summaries are extracted from the Packet Counts Tables as Packet Counts Objects, one for each analysis window within the analysis period. Each Packet Counts Object consists of a set of category attributes and a set of associated summary attributes. As it is expected that a Packet Counts Object may exist in an environment independently from the Packet Counts Table from which it is retrieved, all implicit categories of the table are explicitly defined for the Packet Counts Objects.

An instantiation of a Packet Counts Object, \( \rho \), may therefore be represented as

\[
\rho = (t_s, win, tid, n_t, n_r, n_l, n_d) \tag{5.2}
\]

A query specification, \( Q \), requesting packet counts summary for a specific time period may be represented as

\[
Q = (t_1, t_2, tid, win) \tag{5.3}
\]

The response to this query may be represented is a set of Packet Counts Objects, \( \mathcal{R} \) representing the summaries for analysis windows within the specified time period.

\[
\mathcal{R} = \{\rho_i | 1 \leq i \leq n\} \tag{5.4}
\]

where

\[
n = \frac{Q[t_2] - Q[t_1]}{win}
\]

As mentioned previously, additivity of packet count summaries allows generation of summaries for larger analysis window sizes from summaries of smaller, disjoint analysis windows that exist for the period of the larger analysis windows. The following represents the operations required to generate a set, \( \mathcal{R}' \), of Packet Count Objects, \( \rho' \), for a window size \( win' \) such that \( win \) is an integer fraction of \( win' \) and \( \rho'_j[t_s] = (j-1) \times win' + \rho_j[t_s] \) where \( 1 \leq j \leq m \) and \( m = \frac{\frac{n \times win}{win'}}{win'} = \frac{Q[t_2] - Q[t_1]}{win'} \).
### 5.4 Calculation of Packet Counts Summaries from the Primitive Database

Calculation of packet counts summaries from the primitive database, for a specific analysis period, involves reading the transmitted and the received data tables simultaneously and counting:

- packets logged in the transmitted data table
- packets logged in the received data table
- events indicating loss of packets in the received data table
- events indicating duplication of packets in the received data table

The **PacketCounts** object represents a statistical object that contains packet counts summary data as well as the appropriate category attributes values. Upon instantiation, this object receives an object of the **AggregateDelayQuerySpecs** class, the number of transmitted packets, the number of received packets and the number of packets lost and duplicated during communication. An object of the

\[
\rho'_j[n_i] = \sum_{k=(j-1)\times\frac{\text{win}'}{\text{win}}}^{k\times\frac{\text{win}'}{\text{win}}+1} \rho_k[n_i] 
\]

\[
\rho'_j[\rho_i] = \sum_{k=(j-1)\times\frac{\text{win}'}{\text{win}}}^{k\times\frac{\text{win}'}{\text{win}}+1} \rho_k[\rho_i] 
\]

\[
\rho'_j[n_d] = \sum_{k=(j-1)\times\frac{\text{win}'}{\text{win}}}^{k\times\frac{\text{win}'}{\text{win}}+1} \rho_k[n_d] 
\]

A counts entity, \( e' \) for each Packet Counts Object in \( S' \) may be represented as

\[
e'(\rho'_j[t],\rho'_j[n_i],\rho'_j[n_r],\rho'_j[n_l],\rho'_j[n_d])
\]
AggregateDelayQuerySpecs class represents the category attributes for the specific instantiation of the PacketCounts object. This object contains the start and end times for which the PacketCounts object has been derived, the test identifier and the analysis window size.

The remaining instance variables, i.e. variables representing different counts, represent the summary attributes for the PacketCounts object. Instantiation of a PacketCounts object may throw an IllegalQueryException if an error occurs whilst attempting to instantiate the object of AggregateDelayQuerySpecs class. Alternately a blank PacketCounts object may be instantiated.

The PacketCounts object contains public methods that provide read access to the category attribute object as well as each of the summary attribute values.

Objects of the PacketCounter class receive two objects of the TestPacket class, one from the transmitted data table and other from the received data table. Appropriate methods within the PacketCounter object attempt to determine whether a condition of packet loss or duplication has occurred. If a condition of packet loss has occurred, a LostPacketException is thrown. Alternately if a condition of packet duplication has occurred, a DuplicatedPacketException is thrown.

The PacketCounter object maintains a count of the number of transmitted test packets logged in the transmitted data table and the number of received test packets logged in the received data table as well as the number of test packets lost and duplicated during communication. These counts are maintained since the instantiation of the object.

Method countPackets() receives two objects of the TestPacket class, one from the transmitted data table and the other from the received data table. It throws a LostPacketException or a DuplicatedPacketException if either event occurs. During the process it increments different instance variables representing various summaries for the Packet Counts data.

Another method resetCounter() may be used to initialise all the instance variables to zero. Appropriate public methods also provide read access to the respective instance variables.
A `PacketCountsCalculator` object derives the Packet Counts summaries from the primitive database tables for a specific analysis period. `PacketCountsCalculator` objects contain the following objects:

- two objects of the `RawDB` class, one to manage the transmitted data table and the other to manage the received data table.
- an object of the `QueryResponse` class which provides methods to retrieve `TestPacket` objects from the objects of the `RawDB` class.
- an object of the `PacketCounter` class, which is used to maintain a count of transmitted, received, lost and duplicated packets as the `TestPacket` objects are retrieved from respective data tables.

The structure of the `PacketCountsCalculator` class is shown in figure 5-2.

The constructor method of the `PacketCountsCalculator` class receives a query specification of the `AggregateDelayQuerySpecs` class from the client or the higher level object. It also receives a string representing the path to the database directory. The constructor instantiates different objects contained within the `PacketCountsCalculator` object and triggers the `countPackets()` method.
The `countPackets()` method then reads the `TestPacket` objects from both the `RawDB` objects simultaneously via the `QueryResponse` object's `getNext()` method till `QueryEndExceptions` are thrown by both the `RawDB` objects. The `countPackets()` method then passes these `TestPacket` objects to the `PacketCounter` object. If the `PacketCounter` object throws a `LostPacketException` or a `DuplicatedPacketException`, the `countPackets()` method manages the pointer of the respective database tables appropriately (figure 5-3).

If the end of one of the tables is reached, or the end of query is encountered, a representative dummy test packet, received or transmitted, is generated with the transmit time greater than the end time of query specification. This allows the determination of loss and duplication events at the end of the dataset being processed. A public method, `getPacketCounts()`, retrieves the summaries from the `PacketCounter` and generates an appropriate `PacketCounts` object. This `PacketCounts` object is returned to the client object.

**5.5 Management of Packet Counts Summaries**

Management of packet counts summaries is carried out by an object of the `PacketCountsManager` class. An object of this class receives an object of the `AggregateDelayQuerySpecs` class as the query specification from the client object upon instantiation. `PacketCounts` object for each analysis window within the period of the query specification are generated by triggering the `getNext()` method in the `PacketCountsManager`. This method throws a `QueryEndException` once retrieval of a `PacketCounts` object is attempted beyond the analysis period specified in the query specification.
In addition to an `AggregateDelayQuerySpecs` object, the `PacketCountsManager` receives a string specifying the path to the database directory. The `AggregateDelayQuerySpecs` object represents the query specification and contains
- start and end times of the analysis period
- the analysis window size
- test identifier

The `PacketCountsManager` object uses a `QueryDecomposer` object to decompose a query of the `AggregateDelayQuerySpecs` class into its component queries. Each component query is also an object of the `AggregateDelayQuerySpecs` class.

A constructor for the `QueryDecomposer` class receives the query specification of the `AggregateDelayQuerySpecs` class and decomposes it into component queries corresponding to the window size specified in the composite query. A private method, `checkQueryConsistency()`, checks the consistency of the composite query passed to it. The following checks are made to verify the consistency of the query:
- The entire period of the query can be decomposed into respective component queries
- The window size specified is an integer fraction of 24

If the composite query fails the consistency check, the `checkQueryConsistency()` method throws an `IllegalQueryException` to the constructor. This exception is thrown by the constructor to the client object.

An alternate constructor allows the decomposition of the composite query on the basis of a different window size, as long as the composite query and the window size required pass the above mentioned consistency checks.

A trigger to the `getNext()` method returns the appropriate component query in order. It throws the `QueryEndException` once a component query is accessed beyond the period specified in the composite query.

An object of the `PacketCountsFiler` class in an object of the `PacketCountsManager` class manages the Packet Counts Tables. The database files are maintained in a directory structure as shown in figure 5-4. Analysis data
for each month (primitive as well as derived) are maintained in appropriate files under directories that indicate months of a year (a higher level directory within the database directory).

The Packet Counts Tables are managed physically as files which are initialised for the whole month for a particular test identifier and window size. This allows for efficient access to a particular entry using the following hash function

\[
\text{Location} = \text{TupleLength} \times \left( \frac{\text{HoursFromStart}}{\text{win}} \right)
\]  

where

\[
\text{HoursFromStart} = (\text{DayOfMonth} - 1) \times 24 + \text{HourOfDay}
\]

Upon instantiation, the \texttt{PacketCountsFiler} object receives a string specifying the path to the database directory. The destructor closes the files opened by the \texttt{PacketCountsFiler} object.

Two public methods allow access to the appropriate Packet Counts Tables. The \texttt{getCounts()} method receives an object of the \texttt{AggregateDelayQuerySpecs} class specifying an appropriate component query. This method returns an object of the \texttt{PacketCounts} class after retrieving the data from an appropriate file. Alternately it throws a \texttt{BlankCountsException} if the number of transmitted packets in the \texttt{PacketCounts} object is equal to zero. This exception is an indication that relevant summaries may not exist in the Packet Counts Table and the client application (an object of the \texttt{PacketCountsManager} class in this case) should attempt to derive these summaries from primitive data.

\[\text{Figure 5-4 : Directory Structure}\]
The *getCounts()* method, upon being triggered, attempts to open the appropriate Packet Counts Table. If such a file does not exist, it is initialised (i.e. created) for the specified window size and test identifier. All summary attributes for each category value in the table are initialised to zero. A *BlankCountsException* is thrown to inform the client object that the requested summaries do not exist and have to be derived from the primitive database.

Client applications can trigger the *addCounts()* method to save the packet counts summaries in the appropriate Packet Counts Table. This method throws an *IOException* in case an error occurs during the file I/O operations.

The *PacketCountsManager* object attempts to read packet counts summaries from the Packet Counts Tables with the help of the *PacketCountsFiler* object. If it catches a *BlankCountsException*, it attempts to construct these summaries from the appropriate Packet Counts Tables maintained at the finest granularity (i.e. window size of one hour). This summary construction uses another object of the *QueryDecomposer* class to further decompose the component query being analysed to respective component queries with window sizes of one hour each.

If a *BlankCountsException* is caught during the summary construction process, the *PacketCountsManager* uses an object of the *PacketCountsCalculator* class to derive the packet counts summaries for the specific analysis window of one hour. Once these summaries have been derived, the *PacketCountsFiler*’s *addCounts()* method is used to save these summaries appropriately.

Packet counts summaries derived at the finest granularity may be added together to generate packet counts summaries at the desired level of granularity. The *addCounts()* method of *PacketCountsFiler* is then used to save these summaries generated at a coarser level of granularity.

The architecture of the *PacketCountsManager* class is shown in figure 5-5.

### 5.6 Summary and Discussion

Packet counts summaries provide information on
- The number of test packets transmitted by the transmitting test station within an analysis window for a particular test identifier
• The number of test packets received by the receiving test station within an analysis window for a particular test identifier
• The number of test packets lost within an analysis window for a particular test identifier
• The number of test packets duplicated within an analysis window for a particular test identifier

Figure 5-5: PacketCountsManager Class

The proportion of test packets lost or duplicated with respect to the number of test packets transmitted can provide a strong indication of network performance. Excessive loss of test packets may indicate a condition of congestion or network failure. On the other hand duplication of test packets may indicate problems in the lower layer protocols or the routing algorithms used by the node processors.

Network performance data, including the packet count summaries are processed for specific analysis windows within the analysis period. Due to the additive nature of these summaries, these may be generated for larger analysis windows by
simply adding packet count summaries pre-calculated for smaller window sizes for the same analysis period.

Packet counts summaries are stored as tables (Packet Counts Tables), one for each specific test identifier and window size. Test identifier and window size are treated as implicit category attributes where as the start time of each analysis window acts as the explicit category attribute. Implicit category attributes are not used for classification of statistical entities within the Packet Counts Table.

Packet counts summaries are retrieved from the Packet Counts Tables as Packet Counts Objects in response to queries posted by the client object or any other higher level object. As the Packet Counts Objects are expected to exist in an environment independently of the Packet Counts Tables, implicit category attributes of the Packet Counts Tables are explicitly specified for each Packet Counts Object.

An object oriented system has been described in this chapter to calculate and manage the summaries indicating the amount of test packets transmitted, received, lost and duplicated during the intrusive network monitoring process. This system initialises a fixed sized file for a specific test identifier and window size. The size of the file is a function of the window size and the days in month. All packet counts summaries calculated for this specific query are inserted into the Packet Counts Table for subsequent reuse.

The system attempts to calculate packet counts summaries for a specific analysis window by adding packet counts summaries generated for the period of the queried analysis window on an hourly basis (i.e. window size of one hour). If packet counts summaries for this period are not available on an hourly basis, then the system attempts to compute these from the primitive monitoring database. Once these summaries have been computed on an hourly basis, these may be cached in Packet Counts Tables for the respective test identifier and analysis window size of one hour.
Chapter 6
A Server for Network Performance Information

6.1 Introduction
The Client Server model is the most widely used model for developing distributed systems. Servers are managers of computing resources whereas clients are the users of resources managed by them. Resources commonly managed by servers include databases, disk drives, printers and communication devices such as modems and fax machines.

Client Server database technology aims at managing the data and information as a resource that may be required by multiple clients for different applications. Servers store data and perform some application logic. The client software, usually running on separate hardware, formulates requests for data from the server. These requests are passed to server software via layers of network software.

This chapter describes a server that has been developed to provide information derived from network performance data. These data have been collected by intrusively monitoring the network links under test. The server can accept queries for information at three levels, highly processed information, Intermediate Information and primitive data. Information objects derived as a result of these queries are appropriately packaged and transmitted across the network to the remote client.

Different types of query objects that can be generated by the clients and the relevant information objects derived by the server in response to these objects are explained before discussing the server architecture. This chapter is concluded after explaining an example client that is used to access data from the server.

6.2 Query and Information Objects
The information processing objects used by the server decompose the queries into sub-queries for the required analysis windows within the specified period. The information processing objects then generate information elements for each particular analysis window.

The server architecture (described in the next section) is based on a concept whereby individual information elements of successive analysis windows are
pulled off from the server by the clients. This approach is contrary to a more conventional approach where the complete result of a query is passed onto the client by the server. It is advantageous in scenarios where the server is required to provide significantly large amounts of data that have been derived after extensive calculations. For example, a separate thread on the client may start processing these partial results of a query while the server derives the information for the next component of the query. Moreover if the server crashes or the client is disconnected from the server at any stage, the results of the sub-queries that have been calculated prior to the disconnection or failure have been communicated to the client. If the client caches this information, it need only query the remaining information when the server recovers or is reconnected to the client.

This section describes the queries that a client can make to the server and the results generated by the server as a result of these queries.

6.2.1 Query Objects

The client can issue three different types of queries to the server
- Queries of the QuerySpecification class
- Queries of the AggregateDelayQuerySpecs class
- Queries of the FinalResultsQuerySpecs class

StoreQuery is the super-class of these query objects. Each query object is received by the server as an object of the StoreQuery class. The server can then determine whether this query object is an instance of QuerySpecification, AggregateDelayQuerySpecs or FinalResultsQuerySpecs class. The inheritance hierarchy of these classes is shown in figure 6-1.

Each of the query specification subclasses are described in the following paragraphs

6.2.1.1 QuerySpecification Class

Objects of the QuerySpecification class specify queries to access primitive database. These queries request primitive data of a particular test identifier for a specified period. The queried period is specified by the start and end times of the period.
6.2.1.2 AggregateDelayQuerySpecs Class
Objects of the AggregateDelayQuerySpecs class specify queries for Intermediate Information elements, i.e. objects of DelayAggregate class and PacketCounts class. This class extends the QuerySpecification class and therefore inherits its instance variables and behaviour. In addition it requires the analysis window size to be specified, as the Intermediate Information elements are calculated for different analysis windows within the queried period.

6.2.1.3 FinalResultsQuerySpecs Class
The FinalResultsQuerySpecs class extends the AggregateDelayQuerySpecs class and is used as a query specification for accessing final results from the server. This class essentially has the same functionality as the AggregateDelayQuerySpecs class. It is used to differentiate a request for Intermediate Information from the request for final results at the server.

![Query Object Classes](image)

Figure 6-1: Query Object Classes

6.2.2 Information Objects
The clients can access three types of information elements from the server

- primitive data, i.e., transmitted and received packet data in the form of TestPacket objects, packaged as an object of PrimitiveData2 class.
- Intermediate Information objects of DerivedResults class that contain the information extracted from objects of the DelayAggregate and the PacketCounts classes for a particular analysis window
- Final results objects of FinalResults class that contain average delays in percentiles that are multiples of 5, an object of the DelayDistribution class and an object of the PacketCounts class for a particular analysis window
**Results** is the super-class for the above mentioned classes. The client receives the results of a query from the server as an object of **Results** class. The client can perform an appropriate action after determining the actual class of the received object.

The inheritance hierarchy of the results objects generated by the server is shown in figure 6-2.

Subclasses of **Results** class are explained in the following paragraphs.

### 6.2.2.1 PrimitiveData2 Class

Primitive data are accessed from the server in pages, where each page contains details of transmitted and received test packets for one hour of monitoring. Thus each request for primitive data is ceiling to retrieve data as pages. For example, if data are requested for a period between 1996/11/1/11:59:20 and 1996/11/1/12:00:02, the pages returned by the server will contain data monitored between 1996/11/1/11:00:00 and 1996/11/1/13:00:00. Thus two page objects are returned, one containing primitive data monitored between 1996/11/1/11:00:00 and 1996/11/1/11:59:59 and the other containing primitive data monitored between 1996/11/1/12:00:00 and 1996/11/1/12:59:59.

![Diagram](image)

Figure 6-2 : Classes to Encapsulate Network Performance Information

**PrimitiveData2** class is used to package each page of primitive data (transmitted and received packets). Objects of **PrimitiveData2** class are prepared by the server in response to the query of the **QuerySpecification** class from the client and are passed to the client.
The **PrimitiveData2** class instantiates two objects of **Vector** class, i.e. the **TransmittedData** and **ReceivedData** objects. **Vector** class implements a growable array of objects. Like any array, it contains components that can be accessed using an integer index. However the size of a **Vector** can grow or shrink as needed to accommodate adding or removing of items after the **Vector** has been created.

An object, **QuerySpecs**, of the **QuerySpecification** class is instantiated at construction and provides information regarding the primitive data page contained in the specific instance of the **PrimitiveData2** object.

The **TransmittedData** object contains **TestPacket** objects retrieved from the transmitted data table which have their transmit times falling between the start and end times of the **QuerySpecs** object. Similarly the **ReceivedData** object contains **TestPacket** objects retrieved from the received data table which have their transmit times falling between the start and end times of the **QuerySpecs** object.

`getReceiveCount()` and `getTransmitCount()` methods provide the number of received and transmitted **TestPacket** objects packaged in that instantiation of **PrimitiveData2** class. Similarly each trigger to `getTransmittedPacket()` and `getReceivedPacket()` returns a **TestPacket** object representing a transmitted and a received test packet from the **TransmittedData** and **ReceivedData** objects respectively. Each trigger to these methods increments **RetrieveCounterRx** and **RetrieveCounterTx** integers which act as indices to the **ReceivedData** and **TransmittedData** objects respectively. These methods throw **QueryEndException** once they are triggered and the value of either **RetrieveCounterRx** or **RetrieveCounterTx** is more than or equal to the number of **TestPacket** objects in the **ReceivedData** or **TransmittedData** objects respectively.

**TestPacket** objects in the **ReceivedData** and **TransmittedData** objects can also be retrieved randomly. This is accomplished by setting the **RetrieveCounterRx** and **RetrieveCounterTx** integers respectively. Triggering the `setTxIndex()` and the `setRxIndex()` methods will set the **RetrieveCounterTx** and **RetrieveCounterRx** integers equal to the parameters passed to these methods.
TestPacket objects representing the received and the transmitted test packets may be added to the ReceivedData and TransmittedData objects by passing them as parameters to the addReceivedPacket() and addTransmittedPacket() methods respectively.

6.2.2.2 DerivedResults Class
Clients may request information in the form of Intermediate Information so that it may be possible for clients to re-cycle and reuse this Intermediate Information. The server provides Intermediate Information as objects of the DerivedResults class. These objects are generated for specific analysis windows. These objects encapsulate objects of the PacketCounts class and only the sorted delay values from the objects of the DelayAggregate class for the specified analysis windows. Objects of the DerivedResults class are created by an object of AggregateDataOptimiser class.

The structure of AggregateDataOptimiser class is shown in figure 6-3.

![Figure 6-3: An Object of AggregateDataOptimiser Class](image)

Objects of the DerivedResults class are generated by passing an object of the DelayAggregate class and an associated object of the PacketCounts class as parameters to getDerivedResults() method of an object of AggregateDataOptimiser class.

An object of the DerivedResults class encapsulates the sorted delay values of the object of the DelayAggregate class as a Vector object. This optimises the handling of sorted delay values, the number of which vary with the analysis window size, for storage as well as communication. The
sorted delay values can be retrieved by triggering the `getSortedDelays()` method.

In addition to the sorted delay values, objects of the `DerivedResults` class also contain the relevant objects of the `PacketCounts` class. An object of the `PacketCounts` class can be included in an object of `DerivedResults` class by passing it as a parameter to the `addPacketCounts()` method. The object of the `PacketCounts` class in an object of the `DerivedResults` class can be retrieved by triggering the `getPacketCounts()` method.

The appropriate object of the `DelayAggregate` class can be extracted by passing the `DerivedResults` object as a parameter to the `getDelayAggregate()` method of an `AggregateDataOptimiser` object.

The object of the `AggregateDelayQuerySpecs` class for a particular `DerivedResults` object may be retrieved by triggering the `getQuerySpecs()` method of the appropriate `DerivedResults` object.

### 6.2.2.3 FinalResults Class

The server can provide the clients with information that is processed further than the Intermediate Information. This information is provided as objects of the `FinalResults` class, that are instantiated for specific analysis windows.

An object of the `FinalResults` class can provide
- an object of the `DelayDistribution` class for the specific analysis window
- an object of the `PacketCounts` class for the specific analysis window
- Average delay values in percentiles that are multiples of 5 for the specific analysis window

Information contained in an object of the `FinalResults` class is derived from information in an object of the `DelayAggregate` class and an object of the `PacketCounts` class for the specified analysis window. Objects of the `FinalResults` class are generated by passing the appropriate objects of the `DelayAggregate` and the `PacketCounts` classes as parameters to the `getFinalResults()` method of an object of `AggregateDataOptimiser` class.
The query specification for a particular object of FinalResults class can be retrieved by triggering the getCategoryAttribute() method of that object. This query specification is returned as an object of the AggregateDelayQuerySpecs class.

An object of the PacketCounts class for a particular object of the FinalResults class can be retrieved by triggering the getPacketCounts() method of that object. An object of the PacketCounts class for a particular object of the FinalResults class can be included in it by passing it as parameter to the addPacketCounts() method of that object of the FinalResults class.

An object of the DelayDistribution class can be added to a particular object of the FinalResults class by passing that object of the DelayDistribution class as parameter to the addDelayDistribution() method of the FinalResults object. An object of the DelayDistribution class can be retrieved by triggering the getDelayDistribution() method of that object of the FinalResults class.

An object of the FinalResults class contains average delay values for specified analysis windows in percentiles of 5. These average delay values are passed as an array of 20 integers as parameters to the addDelays() method. An object of the FinalResults class can provide average delay value for the specific analysis window for percentiles that are multiples of 5. This value can be retrieved by passing the start and the end values of the percentiles as parameters to the getDelays() method. This method throws an InputOutOfRangeExpectation if the start and end values of percentiles are not in order or are not multiples of 5.

6.3 Server Architecture
The server for the network performance information has been implemented as two separate objects. One object is the remote object that actual performs the server operation. The other object acts as the manager for the remote object. This object instantiates the remote object and binds it to the specific port so that prospective clients can access the information provided by the server.

The following paragraphs provide information regarding the remote server object and the server manager:
6.3.1 Remote Server Object

The remote server object uses Java's Remote Method Invocation (RMI) feature. This feature enables a program operating as a client to make method calls on objects located on a remote server machine. StoreIF2 class is the interface for the remote server object and is used to create the skeleton and stub classes.

The server for the network performance information has been implemented as a single threaded entity that operates on a log on / log off principle. Once a client has logged on to the server, no other client can log on to the server and therefore interfere with the server's operation until one of the following conditions occur:

- the client that is being serviced logs off the server
- the server logs the client off due to the occurrence of an error condition

It is possible to implement a multi-threaded server which can accept requests from multiple clients concurrently.

The server processes the queries by breaking these down into sub-queries, one for each analysis window in the queried period for deriving processed data and Intermediate Information or one for extracting each page of primitive data. The client can pull off the results of each sub-query by calling a remote method successively for each sub-query.

The server object of the Store2 class actually consists of a number of objects that process the information requested by the clients and format it appropriately for communication to the client. The following paragraphs briefly describe each of these objects:

6.3.1.1 PrimitiveDataCollector2 Class

PrimitiveDataCollector2 class retrieves the queried primitive data, i.e. the transmitted and received test packets, formats these as pages and returns these pages to the calling object. Each page contains test packets transmitted and received in one hour.

An object of this class is instantiated by passing an object of QuerySpecification class and a string representing a path to the database.
directory to the constructor. The query specified by the object of the QuerySpecification class is ceiled so that appropriate pages may be retrieved from the primitive database. This object is converted to an object of AggregateDelayQuerySpecs class with analysis window size of one hour. The constructor instantiates an object of QueryDecomposer class, PageQueryGenerator, to decompose this object of the AggregateDelayQuerySpecs class into sub-queries for each page (i.e. sub-query period = 1 hour) in the period specified in the query specification.

Each page of primitive data can be retrieved by triggering the getNext() method of an object of the PrimitiveDataCollector2 class. This method returns a page corresponding to the page query obtained by triggering the getNext() method of the PageQueryGenerator object. The getNext() method of an object of the PrimitiveDataCollector2 class throws a QueryEndException at the calling object if data access is attempted beyond the query end time.

Two objects of the RawDB class are instantiated for each page query. The TxTable object provides access to the database table containing the data regarding the transmitted test packets whereas RxTable object provides access to the database table containing the data regarding the received test packets.

An object of QueryResponse class, ResultsGenerator, is instantiated for each page query. This object manages the retrieval of test packet data as TestPacket objects by using the TxTable and RxTable objects. Each TestPacket retrieved is added to an object of PrimitiveData2 class, ExtractedData, which is instantiated for each page query. Once each TestPacket object has been retrieved from the respective database tables for a specific page query and added to the ExtractedData object, ExtractedData object is returned to the calling method.

Method showQuery() lists the values of the instance variables of an object of class QuerySpecification passed to it as a parameter.

The structure of the PrimitiveDataCollector2 class is shown in figure 6-4.
6.3.1.2 DerivedResultsCollector Class

Objects of this class provide objects of DelayAggregate and Packet Counts classes for successive analysis windows within a query period. This class contains two objects, the DelayCalculator object and the CountsCalculator object of the DelayAggregateManager3 and PacketCountsManager classes respectively. An object of the DerivedResultsCollector class can be instantiated by passing an object of the AggregateDelayQuerySpecs class specifying the query, and a string specifying the database path. The constructor instantiates the DelayCalculator and the CountsCalculator objects. Successive objects of DelayAggregate and PacketCounts classes are retrieved by triggering the getNextDelay() and getNextCounts() methods of an object of DerivedResultsCollector class. These methods throw either QueryEndException once they are triggered beyond the end of query or IOException when an error occurs while reading relevant data files.

![Diagram of DerivedResultsCollector Class](image)

Figure 6-4: PrimitiveDataCollector2 Class

The structure of the DerivedResultsCollector class is shown in figure 6-5.

An object of Store2 class also uses an object of the AggregateDataOptimiser class to generate the objects of DerivedResults and FinalResults classes respectively.
An object of **Store2** class is instantiated by passing the database path to the constructor method as a string object. Any client wishing to access data and information maintained by the remote server must log in to the server. This is accomplished by remotely invoking the *login()* method and passing an object of the **StoreQuery** class as a parameter, specifying the request.

The server checks whether it is already serving any other client or not. If it is serving another client, it throws a **ServerBusyException**. If it is free, then it logs the client in and returns a login identifier (login ID). This is actually a long integer number that represents the time when the client logged into the server. All other remote method invocations will require the client to identify itself by passing its login ID as a parameter in these remote method calls. This prohibits any other client from interfering in the server operation to serve a logged on client.

Once a client logs in, the server instantiates the appropriate data collector objects, i.e. of the **PrimitiveDataCollector2** class or the **DerivedResultsCollector** class. The data collector instantiated depends upon the actual subclass of the query specification object, i.e. of the **StoreQuery** class, which is passed as a parameter to the *login()* method. Thus if the actual subclass is **QuerySpecification**, then **PacketsCollector** object of the **PrimitiveDataCollector2** class is instantiated. On the contrary if the actual subclass is either...
AggregateDelayQuerySpecs or FinalResultsQuerySpecs, ResultsCollector object of the DerivedResultsCollector class is instantiated.

The client may then start accessing the requested information by triggering the getNext() method of the Store2 object. The client has to pass its login ID as a parameter to this method each time it is triggered. If the login ID passed by the client does not match with the login ID maintained by the server, the server throws the WrongIDException. If the client has passed the correct login ID, appropriate methods in the respective collector objects are triggered to provide the requested information (Objects of the PrimitiveData2 class, DerivedResults class or FinalResults class). These result objects are type casted as QueryResults objects of the Results class and returned to the client.

If during the data retrieval operation, an IOException or RemoteException is thrown due to an error, the getNext() method of the server object logs the client out and throws the respective exception to the client. At the end of the query the getNext() method of the Store2 object throws a QueryEndException to the client.

The client may log out from the server by passing its login ID as a parameter while remotely invoking the logout() remote method. If the login ID passed by the client does not match the login ID maintained by the server, this method throws the WrongIDException. If the login ID maintained by the server matches the login ID it receives, it logs the client out by setting the login ID field it maintains to zero and passing it to the client as an acknowledgement.

The structure of the Store2 class is shown in figure 6-6

6.3.2 Server Manager
An object of the StoreManager2 class instantiates a server object, i.e. an object of the Store2 class and binds it to a specified port. An object of this class receives upon instantiation
- the host name for the computer over which the server is implemented
- port number to bind the server
- the database directory

It then generates a suitable URL string based on the specified parameters, instantiates the server and binds it to the specified port.
6.4 Example Client

This section presents a simple example client that may be used to extract information from the server described in this chapter.

This client requires the host and the port number where the server is expected to be bound. The client program presents the user with a simple menu requesting the choice of the type of information required by the user. The user may request retrieval of primitive monitoring data, Intermediate Information or processed information.

Upon specification of the type of required information, the user is requested to enter other query parameters, i.e.

- the start and end times of the query
- the test identifier
- the analysis window size (in case Intermediate and processed information is requested)
The client then constructs a suitable query specification object (QuerySpecification class for accessing primitive data, AggregateDelayQuerySpecs class for Intermediate Information and FinalResultsQuerySpecs class for processed information). These objects are type casted to the StoreQuery class for communication to the server.

The client then attempts to lookup the server. If it is able to find the server at the specified host and port number, it attempts to log in by invoking the login() remote method and passing the query specification object as a parameter. If the server is busy, i.e. it is serving another client, it throws a ServerBusyException. The client, upon catching this exception, returns to the main menu. If the server is free, it logs the client in and returns the login ID to the client. The client can then start retrieving the requested information by invoking the getNext() remote method. Each invocation of this method returns an object of Results class. Depending upon the type of information requested by the client, the Results object may be type casted as an object of PrimitiveData2 class for a particular page of primitive data, an object of DerivedResults class for Intermediate Information of a particular analysis window or an object of FinalResults class for processed information of a particular analysis window.

Once the server reaches the end of the query, the getNext() remote method throws the QueryEndException. The client after catching this exception, should log out of the server by invoking the logout() remote method. The server logs the client out and as an acknowledgement returns a long integer with a value 0. The getNext() and the logout() remote methods require the client to pass the login ID as its identifier. If the client passes a wrong login ID, each of these methods throw the WrongIDException. This prohibits any other client from interfering in the service being provided by the server to the client that is logged in.

Any error that causes the server to throw the IOException or the RemoteException, results in the automatic logging out of the client. This operation is shown in the state transition diagram of figure 6-7.

6.5 Summary

Information and data maintained by an organisation are considered as its resources and may need to be appropriately disseminated or made available to different departments or individuals within the organisation. In the simplest case this is achieved by allowing the users to access the data maintained centrally via remote
terminals. Alternatively, this data may be distributed appropriately in order to fulfill different performance as well as structural requirements.

A single threaded server that has been developed to maintain network performance data is explained in this chapter. These performance data are acquired by intrusively monitoring data communication networks and suitable information is derived from the monitored data. This object oriented system has been developed over Java’s Remote Method Invocation (RMI) facility.

The server operates on the log on / log off principle, where any client requesting data from the server has to log on to the server. The server generates appropriate information objects in response to the query object sent by the client. The server can provide

- primitive data as pages of transmitted and received test packet details (each page contains monitoring information for one hour of monitoring),
- Intermediate Information for specified analysis windows (sorted delay values and the number of test packets transmitted, received, lost and duplicated),
- processed information for specified analysis windows (average delays in percentiles of 5, delay distribution and the number of test packets transmitted, received, lost and duplicated).

Figure 6-7 : Client Operation
The ability of this server to provide performance information processed at different levels allows a variety of clients to acquire the required information. Clients with minimum resources (thin clients) can acquire information as objects of the FinalResults class. These clients do not possess adequate resources to process network performance data and thus depend on the server to appropriately process the required information and provide the final results.

On the contrary, clients with a significant amount of resources (fat clients) can request the server to provide adequate Intermediate Information or primitive data. These clients can cache acquired data and information and process them according to the requirements of the applications running at these clients. Duplication of information processing elements of the server at the clients may be required in such a scenario. However utilising the client's resources to cache and process the extracted data and information may increase server availability.
Conclusions & Directions for Future Research

7.1 Discussions & Conclusions
Intrusive network performance monitoring involves transmitting test packets over a network under test. The delay experienced by these test packets along with the number of test packets lost and duplicated provide information regarding the performance characteristics of the network elements being tested.

Network performance monitoring can generate significantly large amounts of primitive data, which can provide only very limited information regarding the network performance characteristics. These primitive data may need to be processed significantly to derive appropriate summaries regarding network performance.

Performance information for data communication networks is generally reported for specific analysis windows within the analysis period. For example, summaries may be required for a particular week in analysis windows of 3 hours each.

Network performance needs to be analysed in a number of different ways and hence a wide range of summaries may need to be derived. Moreover network performance information may also need to be correlated with the network configuration data as well as the instrumentation logs of the network monitors.

Because of the large volume of data collected by the monitors as well as the diversity of information processing applications, warehousing these data and the derived information can significantly enhance the performance of the information processing applications. Moreover this centralised and integrated approach to information management ensures provision of consistent and accurate information.

Data warehouses are architected to efficiently process queries and provide the information required to assist in the decision making process. In contrast, OLTP (On-Line Transaction Processing) systems are generally optimised to efficiently manage operational database transactions.
Conclusions & Directions for Future Research

Queries posted to a data warehouse generally require summaries derived from historical primitive data. In order to efficiently process these queries, the primitive data are regularly retrieved from the operational databases and integrated into the data warehouse. During the integration process various inconsistencies in the data from different sources are resolved. Moreover summaries are derived and maintained in the warehouse at different levels of granularity so as to satisfy different information requirements.

However it has been argued that it is extremely difficult to model all the summary data that may be required in a specific domain. Thus data warehouses maintain primitive data for significantly long periods of time so that unforeseen information requirements may be fulfilled. Frequent accesses to the primitive data greatly reduces the performance of information systems as well as the applications requesting the desired information.

Reusable components and devices reduce system development costs significantly. Reusability of data and information maintained and processed by information systems can significantly enhance the performance of these systems. Previous research into information reusability has mainly been concentrated towards storing statistical summaries for subsets of data as their respective additive components. If information is required for a specific dataset which is the union of the subsets maintained initially, these additive components are added and the required summaries are derived from these components.

However these approaches are generally restrictive as they can only provide some basic information elements regarding the process being monitored. Moreover variations in information requirements cannot be adequately supported. In some environments, specially research and development, information requirements vary significantly as the researchers attempt to investigate different attributes of the processes under analysis.

It is possible to pre-process the primitive data so as to derive suitable Intermediate Information elements. Data structures used for Intermediate Information should be rich enough to provide significant information regarding the process being analysed. It should also be possible to combine suitable finest granularity Intermediate Information elements to generate a coarser granularity Intermediate Information element. Moreover the process of deriving the required information
elements from Intermediate Information should be more efficient when compared to the process of deriving the same information elements from the primitive data.

This thesis considers the application of this concept to the processing and management of network delay information derived from primitive network performance data. Two data structures have been evaluated as Intermediate Information structures. Elements of one structure contain the packet delay values sorted in increasing order. This Intermediate Information structure is referred to as Sorted Delays. Elements of the second Intermediate Information structure contain the frequency distribution of packet delay values and hence are termed Delay Distributions.

Sorted Delays can provide numerous information elements regarding the delay characteristics of the network being monitored, for example, averages, variances, frequency distributions, percentiles etc. Sorted Delays can provide these information elements with the highest possible level of accuracy. Information systems using Sorted Delays can combine a number of appropriate finest granularity Sorted Delays to provide Sorted Delays at the required level of granularity. As the finest granularity Sorted Delays contain packet delay values sorted in increasing order, the combination operation can exploit this characteristic by using an efficient merge sort algorithm.

Frequency distributions of packet delay values constituting Delay Distributions, along with the corresponding cumulative frequency distributions, can also provide a variety of information elements, for example, averages, variances, percentiles etc. Finest granularity Delay Distributions with the same bin sizes and number of bins may simply be added to provide the Delay Distribution at the required level of granularity.

The derivation of Sorted Delays can be an expensive process. In contrast, derivation of Delay Distributions with larger bin sizes (and therefore fewer bins) can be a relatively inexpensive process. However the accuracy of information derived from a Delay Distribution depends on the bin size. The larger the Delay Distribution (for a specific range of data values), the more accurate the information derived from it. Increasing the size of the Delay Distributions increases the response time of the information system deriving the required information from these Delay Distributions. Thus an information system using Delay Distributions with significantly large number of bins can present reasonably
accurate information but may have a performance lower than that of an information system deriving the same information from Sorted Delays for small analysis windows.

Databases and information systems based on a single data structure and employing a particular access strategy provide optimal performance to some specific database operations. If the user requirements change and different database operations become prominent the existing structure of the information system provides suboptimal overall performance.

It is possible to develop information systems that reorganise their structure in response to varying
- requirements
- state of the information system
- the characteristics of the data, information and the information processing applications

Information processing and analysis activities in a DSS environment are usually iterative in nature. Analysts process the collected data to generate an initial report. This initial report may indicate the occurrence of an abnormal event which may need to be investigated further. The analyst may then zoom into the events detected in the initial report. This may require processing only a reasonable subset of the data used to derive the initial report.

Moreover it has also been argued that the reports in a DSS environment may only be required to a given level of accuracy. Therefore the entire process may be expedited by using data structures and algorithms that allow for the fast derivation of approximate summaries for the initial reports. As the analyst zooms into different events indicated in the initial report, data structures and algorithms which can provide accurate summaries are used.

This approach can be used in deriving packet delay information by processing suitable Intermediate Information. Delay Distributions with larger bins are derived to provide approximate delay information for large query periods. As the size of the query period decreases for subsequent queries, the bin size of the Delay Distributions used as Intermediate Information also decreases. Finally for small query periods the system uses Sorted Delays as Intermediate Information.
This approach, in addition to maintaining extensive redundant information, also requires an expensive query cost evaluation and optimisation process. Alternatively an information system may derive the finest granularity Delay Distributions from corresponding Sorted Delays and then add these to generate coarser granularity Delay Distributions.

This approach results in a higher system response time for individual queries when compared to the approach that uses separate processes to derive and manage Delay Distributions and Sorted Delays. However as the initial larger query affects the common Intermediate Information Base (Sorted Delays), the entire zoom in or the drill down process can be performed efficiently. Moreover less redundant information needs to be maintained and the query cost evaluation and optimisation process is also relatively simpler.

Even for large queries, the system can provide accurate information with reasonably high performance. This is possible when the query period overlaps the query period of some previous query requesting information at the same level of granularity. It may be possible to access and process the existing high accuracy Intermediate Information elements more efficiently when compared to deriving the required lower accuracy Intermediate Information elements.

In certain circumstances, it may be economical to add appropriate coarse granularity Delay Distributions to generate the Delay Distribution at the required level of granularity. The larger the analysis window size of the component Delay Distributions, the fewer the Delay Distributions needed to generate the required Delay Distributions and the higher the efficiency of the process. For large Delay Distributions, Delay Distributions of only a few wide analysis windows can be reused efficiently. The process of reusing the Delay Distributions of the remaining narrower analysis windows is inefficient when compared to deriving the required Delay Distributions from the corresponding finest granularity Sorted Delays. It is also obvious that the efficiency of this process increases with the increase in the number of available component Delay Distributions.

A required Delay Distribution can be derived from a potentially large number of combinations of Sorted Delays and Delay Distributions. Evaluating the cost of each combination and then selecting the most efficient plan is a complex process. A simpler (but suboptimal) approach suggests the use of appropriate Delay Distributions derived for analysis windows of equal sizes. This reduces the
number of combinations of Delay Distributions and Sorted Delays that need to be evaluated to determine an efficient plan.

In addition to the packet delay information, information regarding lost and duplicated packets can be used to determine the performance characteristics of data communication networks. Lost packets indicate either a condition of congestion or failure of network elements. Duplicated packets, on the other hand, indicate faulty lower layer network protocols.

These conditions can be determined from the intrusively monitored network performance data by scanning the tables for the transmitted and the received test packets. Any test packet that is recorded in the table for the transmitted test packets but not recorded in the table for the received test packets is considered to be lost. Two or more test packets recorded in the table for the received test packets with the same transmit time and packet identifier indicate the existence of duplicated test packets.

Packet counts summaries are derived for specific analysis windows by counting the transmitted test packets, received test packets, test packets lost and duplicated as they are communicated across the network. These summaries are stored in hash tables, one initialised for each test identifier and analysis window size.

Packet counts summaries are retrieved from the appropriate hash tables as Packet Counts Objects. Due to their additive nature, appropriate packet counts summaries can be conveniently added to provide packet counts summaries at the required levels of granularity.

This thesis describes the application of some of these concepts in the implementation of a simple single threaded server for the network performance information. A client application can log in to the server and request primitive network data, Intermediate Information or processed network performance information. Once the client has logged on to the server, no other client can log on to the server and interfere with its operation.

Primitive data are retrieved as pages of transmitted and received test packets. Each page contain transmitted and received test packet details for one hour of monitoring. The server provides the corresponding packet counts summaries for
Intermediate Information or processed information for each analysis window within the query period.

7.2 Directions for Future Research
This section aims to briefly introduce some of the research issues related to the work discussed in this thesis. Investigation into these issues may provide the knowledge to optimally manage and process Intermediate Information derived from primitive datasets.

7.2.1 Cache Management
Derived data and information are redundant with respect to the primitive data [Inm88]. Similarly, Intermediate Information derived at a specific level of granularity is redundant with respect to Intermediate Information maintained at the finest granularity as well as the primitive data.

The information systems and their components described in the preceding chapters attempt to cache any Intermediate Information element that is derived from primitive data or granular Intermediate Information elements. These Intermediate Information elements can be reused if any query posted to the information system request information at the same granularity.

However this can ultimately result in a phenomenal amount of Intermediate Information occupying valuable disk space. Most of these Intermediate Information elements are never required in future. Thus there exists a need to develop a means to efficiently manage the Intermediate Information Base by retaining the Intermediate Information elements for only the period when they are required.

Significant research exists in the management of resources by operating system. The results of research conducted on page replacement algorithms can provide valuable information regarding management Intermediate Information Base.

Page replacement algorithms attempt to approximate the Optimal Page Replacement algorithm [Tan92], [SilGal95]. This algorithm attempts to achieve optimum performance by replacing the page which is not to be used for the furthest time in the future. This algorithm is not realisable as accurate prediction into the future is not possible.
An algorithm to manage the Intermediate Information Base by deleting unwanted Intermediate Information elements may be termed as an Intermediate Information deletion algorithm. An information system using an optimal Intermediate Information deletion algorithm would ensure low response time of the information system at the cost of specified storage (i.e. by retaining the Intermediate Information elements that may be accessed in the near future).

Intermediate Information at the finest granularity is used for the construction of Intermediate Information at coarser granularity. It is therefore not considered for deletion.

A good Intermediate Information deletion algorithm can be based on the pattern of information requests to the information system. This can be achieved by monitoring the information system and determining maximum periods for which historical information is requested with different levels of granularity.

For example, consider a scenario where finer granularity information is generally requested for smaller periods in recent history and the coarser granularity information is requested for larger periods. Moreover it is expensive to process coarser granularity information. A simple strategy in this case would delete finer granularity Intermediate Information elements, derived from the oldest historical data, that have recently not been requested. The algorithm may need to arrive at a balance between the number of Intermediate Information elements of different granularity that may need to be removed. For example, the Intermediate Information Base contains following Intermediate Information elements

- a. 10 elements with a window size of 24 hours
- b. 30 elements with a window size of 12 hours
- c. 40 elements with a window size of 3 hours

An implementation of the strategy mentioned above may delete the following elements

- a. 2 elements with a window size of 24 hours
- b. 6 elements with a window size of 12 hours
- c. 16 elements with a window size of 3 hours
These are the least recently used elements which correspond to the oldest historical information.

This ratio of different Intermediate Information elements may also depend on the maximum size of the Intermediate Information Base. The size of the Intermediate Information Base can be adjusted dynamically within a specific range by monitoring the queries posted to the information system and statistically analysing the historical periods for which information is requested from the information system.

7.2.2 Pre-emptive Pre-processing
In order to reduce the overall response time of the information system in processing queries, it may be desirable to pre-process Intermediate Information elements at specific levels of granularity before these may be required. These Intermediate Information elements may be derived when primitive data is down-loaded from the operational or data collection systems.

A major issue in this regard includes determining the levels of granularity for which Intermediate Information needs to be pre-processed. Work in this direction can also benefit from previous endeavours in operating systems (i.e., pre-paging, file pre-fetching).

An ideal pre-emptive pre-processing algorithm would attempt to pre-process Intermediate Information only at the levels of granularity for which maximum queries may be submitted to the information system. A very simple approach would be to specify the levels of granularity for which information is required from the system.

But this approach makes the system dependant on the users to specify suitable analysis sizes that should be pre-processed. Alternatively the system may automatically attempt to determine the levels of granularity for which information may be required. This may be accomplished by monitoring the queries posted to the information system. Various parameters of these queries (analysis window sizes, queried period) may be statistically analysed. This analysis may generate an ordered list of analysis window sizes. The analysis window size on the top of the list is the one that is required the most whereas the one at the bottom is the one that is required the least. The system can select
analysis window sizes from the top of the list and pre-process the required Intermediate Information elements.

An algorithm that selects analysis windows to pre-emptively pre-process Intermediate Information elements must exhibit adequate **hysteresis** but at the same time should be sensitive enough to changes in the user requirements. In case the user requirements change and information is requested with different analysis window sizes, the system should be able to adapt to the new requirements within a suitable time period. However this change may be temporary, and the ability of the system to resist, to a certain extent, changing to these new parameter may also be desired.

### 7.2.3 Client Data Caching and Exploitation of Client Resources

Generally client server databases can either be **query shipping** or **data shipping** systems [KosFra95], [FraJK96]. Data shipping allows all operators of a query to be executed at the client machine. This approach exploits the resources of powerful client machines, reduces communications costs in the presence of locality or large query results and improves overall system scalability.

Query shipping allows complete evaluation of queries at the servers. Query shipping reduces communication costs for high selectivity queries and exploits plentiful server resources thereby tolerating low performance client machines.

A new approach, **hybrid shipping**, has also been suggested which combines data and query shipping approaches [KosFra95], [FraJK96]. It has been claimed that hybrid shipping can achieve most efficient execution of a query, even though it is the most difficult policy to optimise.

Client server systems based on Intermediate Information can also employ strategies that can exploit client resources to derive the required information from the cached Intermediate Information elements and primitive data. A possible strategy should generate the Intermediate Information for specific queries at the server only if adequate resources are available at the server. Alternatively the server may communicate primitive data or granular Intermediate Information elements for a particular analysis window to the client. The client generates appropriate Intermediate Information for these windows from the received data and information.
Additionally the servers may be configured only to provide granular Intermediate Information or primitive data. This may reduce the load on the servers thereby increasing their availability. This may also reduce data redundancy at the servers. Information communicated by the servers may be cached at the clients to fulfil subsequent information requirements.
References

A Methodology for Building a Data Warehouse in a Scientific Environment,

[Bit79] Bitner J.R. (1979),
Heuristics that Dynamically Organise Data Structures,

Object Oriented Analysis and Design with Applications,
The Benjamin/Cummings Series in Object Oriented Software Engineering

[Bra82] Bray O.H. (1982),
Distributed Database Management Systems,
Lexington Books

Mining Business Databases,

Using Partitioned Databases for Statistical Data Analysis,

Towards a Unified Theory of Managing Large Networks,
IEEE Spectrum, April 1989, pp 39 - 42

[Cac89] Caceres R. (1989),

162
References

Measurement of Wide Area Network Traffic,
Technical Report CSD-89-550, Computer Science Division,
University of California, Berkeley, USA

Measuring Bottleneck Link Speed in Packet Switched Networks,
Technical Report BU-CS-96-006, Department of Computer
Science, Boston University, Boston, USA

[CCTA90] Central Computer & Telecommunications Agency (1990),
Managing Information as a Resource,
Crown Copyright 1990,
ISBN 0 11 330529 X

[CCTA94] Central Computer & Telecommunications Agency (1994),
Managing Reuse,
Crown Copyright 1994,
ISBN 0 11 330616 4

[CerPel85] Ceri S. & Pelagatti G. (1985),
Distributed Databases, Principles & Systems,

[CheMcN89] Chen M.C. & McNamee L.P. (1989),
On the Data Model & Access Method of Summary Data
Management,
IEEE Transactions on Knowledge and Data
Engineering, Dec 1989, Vol. 1, No. 4, pp 519 - 529

[CheON93] Cheetham R.P., Oomen B.J. & Ng D.T.H. (1993),
Adaptive Structuring of Binary Search Trees Using Conditional
Rotations,
IEEE Transactions on Knowledge & Data Engineering, Vol. 5,
No.4, August 1993

[Coo91] Cooling J.E. (1991),
Software Design for Real Time Systems,
Chapman & Hall, University and Professional Division
References

Distributed Systems: Concepts & Design,
Addison Wesley Publishers & Co.

[Dat95] Date C.J. (1995),
An Introduction to Database Systems,
Addison Wesley Publishing Co.

[Dav78] Davenport R.A. (1978),
Distributed Database Technology - A Survey,
Computer Networks, Vol 2, No. 3, July 1978, pp 105 - 167

Simulation Modeling with Pascal,
Prentice Hall Inc.

[Dea64] Deardan J. (1964),
Can Management Information be Automated?,

A Model of a Self Organising Data Management System,

The Operation of a Model Self Organising Data Management System,

[Dem94] Demarest M. (1994),
Building the Data Mart,
DBMS, July 1994, Vol 7, No. 8, pp 44 - 50

[EinSeg78] Ein-Dor P. & Segev E. (1978),
Managing Management Information Systems,
Lexington Books
*NETMON II, a Monitoring Tool for Distributed and Multi-
processor Systems*,
Data Communication Systems & Their Performance, edited by
G. Pujolle & R. Puigjaner, Elsevier Science Publications, B.V.
(North) Holland

*Information Network, Planning & Design*,
Prentice Hall Inc.

[Fra93] Franklin M.J. (1993),
*Exploiting Client Resources Through Caching*,
5th International Workshop on High Performance Transaction
Systems (HPTS), Asilomar, CA, September 1993

*Performance Trade-offs for Client Server Query Processing*,
ACM SIGMOD International Conference on Management of Data
(SIGMOD 96), Montreal, Canada, June 1996

*Support for Scientific Database Management*,
in Statistical and Scientific Databases edited by Zbigniew
Michalewicz, Ellis Howard Series in Computers and Their
Applications (1991)

*Scientific Database Management : Final Report*,
Technical Report 90-21, Department of Computer Science,
University of Virginia, Charlottesville, VA22903, August 1990

*Bringing User Perspective to Data Warehouses : 21 Points of
Consideration*,
in Building, Using and Managing the Data Warehouse, edited by
Ramon Barquin and Herb Edelstien,
Prentice Hall Inc.
*Information Resource Management for Corporate Decision Support*,

[Ham77] Hammer M. (1977),
*Self Adaptive Automatic Database Design*,

*The Stanford Data Warehousing Project*,
IEEE Data Engineering Bulletin, June 1995

[HanMaier94] Hansan D.M. & Maier D. (1994),
*Using an Object Oriented Database to Encapsulate Heterogeneous Scientific Data Sources*,

[Har97] Harinarayan V. (1997),
*Issues in Interactive Aggregation*,

[Hel92] Held G. (1992),
*Network Management*,
John Wiley & Sons Inc.

[Inm88] Inmon W.H. (1988),
*Information Engineering for the Practitioner: Putting Theory into Practice*,
Yourdon Press Computing Series
References

[Inm96] Inmon W.H. (1996),
*Building the Data Warehouse*,
John Wiley & Sons Inc.

[Inm96b] Inmon W.H. (1996),
*The Data Warehouse & Data Mining*,

[InmWWW] Inmon W.H.,
*Excerpts from What is a Data Mart*,
http://www.d2k.com/d2k/control.cgi?library2

John Wiley & Sons Inc.

*COMan - Coexistence of Object Oriented and Relational Technology*,
Proceedings of the 13th International Conference on Entity Relationship Approach, Manchester, December 1994, pp 259 - 277

*Migration to Object Data Management*,
OOPSLA Workshop on Legacy Systems and Object Technology

*Object Oriented Programming for Structured Procedural Programmers*,
IEEE Computer, October 1995, pp 48 - 57

*Letting the Users Sleep, Part 2: Nine Decisions in the Design of a Data Warehouse*,
DBMS, January 1997
References

*High Performance Relational Database Design*,
Ellis Horward Series in Computers and their Applications

*The Art of Computer Programming, Volume 3, Sorting & Searching*,
Addison Wesley Publishing

*A Study of Query Execution Strategies for Client Server Database Systems*,
Technical Report CS-TR-3512 & UMIACS-TR-95-85,
Department of Computer Science & UMIACS, University of Maryland

[Laf90] Lafore R. (1990),
*The Waite's Group Turbo C Programming for the PC and Turbo C++*,
SAMS Publishing

[Lam95] Lambart M. (1995),
*A Model for Common Operational Statistics, Request For Comments, RFC 1857*,
RFC Archive, Networking Group

[Lam96] Lambart R. (1996),
*Data Warehousing Fundamentals, What You Need to Know to Succeed*,
Data Management Review, March 1996

[Lan85] Land F. (1985),
*Is an Information Theory Enough*,

*Information & Information Systems : Concepts & Perspectives*,
in Information Analysis, Selected Readings edited by Robert Galliers, Addison Wesley Publishing Co.


An Information Systems Manifesto,
Prentice Hall Inc.

Client Server Databases,
Prentice Hall Inc.

[Mea76] Meadow C.T. (1976),
Applied Data Management,
John Wiley & Sons

Maintenance of Data Cubes and Summary Tables in a Warehouse,
SIGMOD 97.

On Behaviour, Inheritance and Evolution,

[Omi96] Omiecinski E. (1996),
Concurrent File Re-organisation : Clustering, Conversion & Maintenance,

Performance Monitoring of Networks : Architecture, Operation and Dissemination,
Presented at the BT's URI (British Telecom's University Research Initiative) Workshop, University College London, April 1997

Performance Measurements of the SMDS,

On the Monitoring and Measurement of Quality of Service of SuperJanet,
References

13th Teletraffic Symposium, Strathclyde, March 1996

[Pre94] Preece J. (1994),
*Human Computer Interaction*,
Addison Wesley Publishing Co.

*Structure & Organisation of Very Large Databases*,
in Critical Factors in Data Management edited by Fred Gruenberger, pp 127 - 146,
Prentice Hall Inc.

[Rab92] Rabie S. (1992),
*Integrated Network Management, Technologies & Implementation Experience*,
IEEE Infocom 92, Conference on Computer Communications, Florance, Italy

*DAD: A Real-Time Expert System for Monitoring of Data Packet Networks*,
IEEE Network, September 1988, pp 29 - 34

[Rad96] Raden N. (1996),
*Multidimensional Data Modeling, Data Model for the Rest of Us*,
Information Week, 29 January 1996

[Raf91] Rafanelli M. (1991),
*Data Models*,
in Statistical & Scientific Databases edited by Zbigniew Michalewicz,
Ellis Howard Series in Computers & their Applications

[RafBT96] Rafanelli M., Bezenchek A. & Tininini L. (1996),
*The Aggregate Data Problem: A System for Their Definition & Management*,
SIGMOD Record Journal, December 1996
*Mefisto: A Functional Model for Statistical Entities*,
IEEE Transactions on Knowledge & Data Engineering, Vol 5., No. 4, Aug 1993, pp 670 - 681

[RicWWW] Richards L.,
*Bringing Performance to Your Data Warehouse*,

[Riv76] Rivest R. (1976),
*On Self Organising Sequential Search Heuristics*,
Communications of the Association for Computing Machinery, February 1976, Vol 19, No.2, pp 63 - 67

[Rop95] Roppel C. (1995),
*In-service Monitoring Techniques for Cell Transfer Delay and Cell Delay Variation in ATM Networks*,
in High Performance Networking, edited by R. Puigjaner, Chapman & Hall

[Row87] Rowley J. (1987),
*What is text retrieval & why is it important*,

*Database Systems Engineering*,
International Thomson Computer Press

*Monitoring Distributed Systems (A Survey)*,
Imperial College Research Report Number DOC 92/93

[Sat91] Sato H. (1991),
*Statistical Models: from a Statistical Table to a Conceptual Approach*,
in Statistical & Scientific Databases edited by Zbigniew Michalewicz,
Ellis Howard Series in Computers & their Applications

Client Server Computing,
Distributed Computing Environments, edited by David Cerutti & Donna Pierson

[Sha75] Shannon R.E. (1975),
Systems Simulation, The Art & the Science,
Prentice Hall Inc.

Properties of Scientific & Statistical Databases,
in Statistical & Scientific Databases edited by Zbigniew Michalewicz,
Ellis Howard Series in Computers and Their Applications

A Layered Approach to Scientific Data Management at Lawerence Berkeley Laboratory,
Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society, March 1993, Vol 16, No. 1 pp 4-8

[Shn92] Shneidermann B. (1992),
Designing the User Interface : Strategies for Effective Human Computer Interaction,
Addison Wesley Publishing Co.

[Sid89] Siddiqui M.H. (1989),
Performance Measurement Methodology for Integrated Services Networks,
PhD Thesis, Loughborough University of Technology

Performance of Local & Remote Bridge Components in Project Unison - 50 Megabits per Second ATM Network,

173


References

Computer Networks, 3ed,
Prentice Hall Inc.

The Object Oriented Design Notation for Software Design
Representation,
IEEE Computer, March 1990, pp50 - 62

Building Executive Information Systems and other Decision
Support Applications,
John Wiley and Sons Inc.

Research Problems in Data Warehousing,
Proceedings of the 4th International Conference on Information and
Knowledge Management (CIKM), November 1995

Database Design,
McGraw Hill Book Co.

Structured Organisation of Clinical Databases,
Proceedings of the AFIPS National Computer Conference, Vol.44,
1975, pp 479 - 485

[Zag97] Zagelow G. (1997),
Data Warehousing - Client Server for the Rest of the Decade,
in Building, Using and Managing the Data Warehouse edited by
Ramon Barquin & Herb Edelstein,
Prentice Hall Inc.
Bibliography


176


Kimbal R. (1996), *Dangerous Preconceptions ; Discovering the Liberating Truths that can lead to a Successful Data Warehouse Project*, DBMS, August 1996


Bibliography


Appendix A

Object Oriented Software Design

A1.0 Introduction
Design methodologies provide developers with tools to design and develop systems at various levels of abstractions. Good design methodologies cater for implementation, subsequent maintenance, potential growth and modification of the systems being developed. Design methodologies also aid in planning and organising these activities. Appropriate use of good design methodologies ultimately ensures suitable reliability as well as good quality of the end product [WasPM90]. Software developers also aim at modularity, designing for reusability and exploiting reusability in the modules that have already been developed and being used in other similar systems.

Conventionally computer software is distributed in two distinct locations of the memory. Code memory or program memory contains lists of instructions that operate on and manipulate data stored in data memory. Traditionally computer programs were developed as lists of instructions that operated on data. Specific portions of programs could be modularised as subroutines, functions or procedures. Functional structuring methodologies have been derived with a premise that a program takes its shape from the functions carried out by the system [Coo91][Som92].

Function oriented design or function structured design methodologies represent a well understood and a widely practised paradigm. Over the past few years, significant advancements have been made in defining suitable design methodologies for function oriented systems. Quite a few CASE (Computer Aided Software Engineering) tools have also been developed to support the design and development process. These tools, depending upon their complexity and functionality, automate the design, development and coding process at various levels.

Booch defines object oriented design as a method of design encompassing the process of object oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design [Boo94]. Object oriented design attempts to model the real world systems more closely than function oriented design [Coo91]. Object oriented design methods
model systems as a collection of objects rather than functions [Som92][KhaAG95].

An object has a state, a behaviour and an identity. The structure and behaviour of similar objects are defined in their common class. The state of an object encompasses all of the (usually static) properties of the object and the current (usually dynamic) values of each of these properties. Behaviour of an object defines its actions and reactions in terms of its state changes. The behaviour of an object represents its outwardly visible and testable activity. Object identity is the property by which it is distinguished from all other objects [Boo94].

Thus all real world entities or objects consist of certain functional components that may be used to vary the internal state of the object. An object may operate on another object to change its state and it may also generate new objects. These interactions between various objects occur through well defined interfaces. The internal structure and the state of an object is hidden from the outside world except for the interfaces through which it is allowed to communicate with other objects [Laf90][Boo94].

Other features of object oriented systems include inheritance and polymorphism. Inheritance is the capability to define a new object class in terms of an existing class. Inheritance aims at developing a generic base class and then using this base class for constructing other application specific classes [KhaAG95][Boo94]. Inheritance is generalisation/specialisation hierarchy. Superclasses represent generalised abstractions and subclasses represent specialisations where fields and methods from the superclasses are added, modified or even hidden [Boo94].

Murray & Robson suggest that using inheritance for implementation reuse causes more long term problems than it solves. They strongly suggest using only behavioural inheritance and organising class structures to describe a genealogy of behaviour. Any implementation reuse is to be considered a beneficial side effect [MurRob95].

Wassermann et al define multiple inheritance as a relationship between classes whereby one class acquires the structure of other classes in a lattice with multiple parents [WasPM90]. However object oriented programming languages supporting multiple inheritance must resolve the following two issues,

- clashes among names from different superclasses
• repeated inheritance

Clashes will occur when two or more superclasses provide a field or operation with the same name or signature as a peer superclass. Repeated inheritance occurs when two or more peer superclasses share a common superclass. In such a situation the inheritance lattice will be diamond shaped. The main issue in this case is whether the leaf classes have one copy or multiple copies of the structure of the shared superclass [Boo94].

**Polymorphism** is the ability of an entity to refer, at runtime, to instances of various classes. Hence the actual operations performed on the receipt of a message depend on the class of instance [WasPM90].

A number of object oriented design methods have been developed to assist software designers in building object oriented software systems. Some of these methodologies are closely linked to the implementation aspects of some object oriented programming languages. Some of these methods emphasise or de-emphasise different aspects of the design process. Almost all of these methodologies, however, maintain the need to model different attributes and components of the system being designed.

This appendix describes the object oriented design notations that have been used in developing the software described in this thesis. The notations described here are a small subset of the notations used by a number of design methodologies.

**A2.0 Developing Object Oriented Systems**

Khan et al suggest following steps for developing object oriented software [KhaAG95]

• statement of the problem (i.e. user requirements) in a simple descriptive language
• identification of object classes, their attributes (data members) and the data operations (member functions) associated with each object class
• decomposition of system specification into a number of modules with each module consisting of one or more object classes
• declaration and definition of each object class by encapsulating its attributes and operations
• identification of message passing between interacting object classes through the requests answered and services required by each object class
• identification of inheritance relations and class hierarchies on the basis of dependencies between object classes
• creation of a logical object oriented model for the proposed system that shows interaction between objects.
• development of algorithms for membership functions of each object class to process its data members
• selection of an appropriate programming language and a suitable implementation strategy
• preparation of test, delivery and maintenance plans

It is known that a software based system cannot be completely described at the design stage entirely on the basis of one aspect [Boo94]. Behavioural description of the system may overlook its architectural characteristics. Similarly the architectural characteristics may not provide any clues regarding the behaviour of the system. Additionally different implementation aspects of the system being designed may need to be modelled appropriately. Thus system description may need to employ a number of different techniques, each providing a suitable representation of the system at the design stage.

For example Booch identifies a set of models that have been found to be semantically rich and expressive. These allow the designer to capture all the interesting strategic and tactical decisions that must be made during the analysis of a system and the formulation of its architecture [Boo94] (Figure A-1)
The *logical view* of a system describes the existence and meaning of the key abstractions and mechanism that form the problem space or define the system architecture. On the other hand, the *physical model* of the system describes the concrete software and hardware composition of the system's context or implementation.

*Object diagrams* provide descriptions of the scenarios in the logical model. *Class diagrams* are used to capture the abstraction of these objects in terms of their common roles and responsibilities. Additionally, it is also desirable to show the allocation of classes and objects to modules in the physical design of a system. This is achieved by means of the *module diagrams*. A module diagram indicates the physical layering and partitioning of a system. *Process diagrams* show the allocation of processes to processors in the physical design of the system. A single process diagram represents a view into the process structure of a system. Process diagrams are used to indicate physical collection of processors and devices that serve as the platform for the execution of the programs being developed. These diagrams also include the connections between different processors and devices by means of which these components communicate with one another.

The diagrams described above are static. They fail to represent the events occurring in the systems. Thus some means need to be employed to express the dynamic semantics of a problem and its implementation. Booch suggests the use of *state transition diagrams* to indicate the event ordered behaviour of the class's instances. *Interaction diagrams* may be used to show the time or event ordering of messages as these are evaluated [Boo94].

Booch also suggests that the fact that these notations are detailed does not mean that every aspect of these must be used at all times. A proper subset of these notations is sufficient to express the semantics of a large proportion of analysis and design issues [Boo94].

**A3.0 Models for Object Oriented Systems**

This section explains some methods used to design the object oriented software described in this thesis. Models for designing and representing object oriented systems are described here with the help of an example. The example attempts to explain the design of software for a timer.
A3.1 Class and Object Identification

Class and object identification are probably the most difficult parts of object oriented design. These also involve identification of various attributes and operations associated with each object and class [Som92].

At the higher level of design, various object requirements and generations also need to be identified. Requirements basically identify objects that are required by another object to perform its operations. Generations, on the other hand, are objects that are generated or produced as a result of the operations an object performs. These may also include exceptions that are thrown by the object on the occurrence of a particular abnormal event as a result of the subject operations. These exceptions may be caught by the client objects so that appropriate operations may be performed.

One of the simplest and oldest methods for object identification is exploiting the natural language description of the systems at various levels of abstractions [Som92]. The basic language analysis suggested by Abott in 1983 starts by identifying key nouns and verbs in system description. Nouns represent the objects whereas verbs may define operations. However this basic approach may need to be refined to achieve some desirable characteristics. For example, the designer may need to generate such descriptions iteratively till a significantly refined description is obtained. Thus any description of the system will have to be analysed with a desirable amount of common sense in addition to the designer's knowledge of the system being designed [Som92]. This description may then provide the starting point for the design.

Consider the example of the timer system. It may be described as follows,

A timer measures time and date which is displayed on the control panel and which may be changed from the control panel. If the value of time and date object required to be set on the clock is incorrect (for e.g., 25 hours or 61 minutes etc.) the clock throws an illegal time exception to the control panel.

This description identifies a clock object which performs the operation of measuring time and date. Time and date are considered as a single object. The time and date object may be changed from the control panel object. If the time
and date value required to be set from the control panel is incorrect, the clock object throws an illegal time exception to the control panel object.

The control panel object may further be described as follows,

The keypad of the control panel generates the time and date to be set on the clock. The display of the control panel shows the time and date measured by the clock. The control panel catches illegal time exception thrown by the clock which is displayed on the control panel's display.

This description identifies the generation from the keypad object in the control panel object. This is the time and date object that contains the time and date values required to be set on the clock object. Similarly this description also identifies the generations from the clock object, i.e. the measured time and date object and the illegal time exception. These objects are the requirements for the display object of the control panel object.

The clock object may further be described as follows,

The clock consists of counters that increment current time and date appropriately. The clock receives the new time and date, which is checked by the time date checker for correctness. If the time and date are incorrect, the time and date checker generates the illegal time exception which is thrown by the clock at the control panel. If the time and date are correct, the time and date are passed to the counters to adjust the current time and date appropriately.

This description manages to identify the objects that clock object may require from control panel object.

These descriptions also identify objects within clock and control panel objects.

Booch describes a number of other techniques that may be used to discover objects and classes that can be used to solve a particular problem. These include the classical approaches, behaviour analysis, domain analysis and structured analysis etc. [Boo94].
A3.2 Inheritance Relationships Between Classes
After appropriate objects have been identified within a system or a subsystem, their relationships in terms of inheritance have to be identified. Inheritance should be implemented in a manner so as to provide specialised versions of generalised classes. For example a generalised class display may have descendants as specialised classes of CRT display, Liquid Crystal Display and LED display (Figure A-2). These specialised classes inherit the generic functionality of the class display. Figure A-2 show classes colour CRT display and monochrome CRT display as descendants of the class CRT display. In this case either of these classes inherit the generic functionality of the class CRT display as well as display.

![Inheritance Relationships Between Classes](image)

A3.3 Architectural Description of an Object Oriented System
System architecture can be represented conveniently by a set of notations. These notations identify the interacting objects in an object oriented system as well as the relation between these objects (Figure A-3).

An object is represented by a circle which encloses the data and operations/functions (methods) internal to the object. The rectangles shown at the boundary represent the interface the object has for interaction with other objects (public methods). Rectangles inside the boundary of the circle represent the functions internal to the object (private methods).

Each object may generate one or more objects as a result of an operation. The flow of objects is shown by a continuous line with the arrow showing the direction of flow. If an object is passed to another object, then the continuous line is drawn from the boundary of the object being passed to the interface of
the object to which it is passed. If an object generates another object then it may be represented by a line from the interface of the object generating the second object to the boundary of the second object.(Figure A-4).

Figure A-3 : Notations for Architectural Design of Object Oriented Systems

The dashed line represents the flow of exceptions i.e. objects that represent the occurrence of abnormal conditions. An object, on detection of an abnormal condition, generates an exception object. This exception object may be returned to the client object so that appropriate action may be taken by the client object.

Continuing the example of a timer, at the highest level, there may only be two objects, control panel and the clock. The control panel may generate an
object, new time and date. This object is required by the clock. If the new time and date is correct the clock sets new time and date. If the new time and date are incorrect, the clock throws an illegal time exception to the control panel. The clock generates the measured time and date which is displayed on the control panel (Figure A-5).

![Diagram](image)

Figure A-5 : Timer System

The control panel consist of a keypad and a display objects. The user may enter, from the keypad, the new time and date value that is required to be set on the clock. The display shows the measured time and date value that is received periodically by the control panel from the clock object. The display also shows the occurrence of illegal time exception, thrown by the clock object if the new time and date set from the keypad are incorrect (Figure A-6).

The clock object of the timer system also consists of two objects. The clock object uses a date and time checker to determine if the value of new time and date are correct and consistent. If the value of new time and date object are incorrect, the date and time checker throws an illegal time exception, which is thrown by the clock object at the control panel object. If the new time and date value are correct, the clock object, sets the counters to the new time and date value. The counters measure the time and date and the measured time and date can be read from the clock object (Figure A-7).

It can be seen that these diagrams show only one public method for the objects described in these examples. These diagrams logically represent the objects constituting a system at an early stage of the design process. It is possible to
refine these descriptions as the design proceeds so that these diagrams represent all or most of the components (public and private methods) of the objects. Thus the system described in figure A-5 can be refined to the description shown in figure A-8.

Figure A-6 : Control Panel Object

Figure A-7 : Clock Object
A3.4 Dynamic Description

The dynamic model of an object oriented system as well as that of a single class can be represented by a state transition diagram. The state of an object represents the cumulative results of its behaviour. At any given point in time, the state of an object encompasses all of its (usually static) properties, together with the current (usually dynamic) values of each of these properties [Boo94].

State transition is the change of state of a system that is caused by the occurrence of an event. A state transition connects two states. A state may have a state transition to itself and it is common to have many different state transitions from the same state. However such transitions must be unique, i.e. there will never be any circumstances that would trigger more than one state transition from the same state.

Each state transition diagram must have exactly one default start state. A state transition diagram may or may not have a stop state.

The state transition diagram for the system shown in figure A-8 is represented in figure A-9. Upon start up, the resetClock() method of the clock object is triggered to reset the clock object. Once the clock object has been reset, the getTime() method of the clock object is triggered. This method returns the MeasuredTime&Date object. This object is passed to the showMeasuredTimeDate() method of the control panel object. This method
when triggered, displays the time and date measured by the clock on the control panel.

![State Transition Diagram of Timer System](image)

Figure A-9 : State Transition Diagram of Timer System

The `getNewTimeDate()` method of the control panel is triggered to determine if the user has entered new time and date values from the control panel's keypad. If the user has not entered any new time and date values on the control panel, `getMeasuredTimeDate()` method is triggered. If the user has entered new time and date values, these are passed as parameter (NewTime&Date object) to the `setTimeDate()` method.

If the time and date values entered by the user are incorrect (for e.g. 25 hours, 61 minutes etc.), the clock object throws an `IllegalTimeException`. This exception is then displayed on the control panel's display by triggering the `showException()` method of the control panel. `getMeasuredTimeDate()` method is then triggered.

**A4.0 Summary**

Notations for the logical design of object oriented software have been described in this appendix. These notations have been used in the design of the software
described in this thesis. A notation attempts to represent the inheritance relationships between different classes that can be used to solve a particular problem. Similarly appropriate notations are used to represent the architectural composition of different objects. These notations also describe the objects required by and the objects generated by the objects being represented. Similarly these notations also describe different objects that internally constitute the object being represented. The dynamic behaviour of the system being designed can be represent by state transition diagrams.
B1.0 Client Server Computing

Client server model is the most widely adopted model for distributed systems. A server is a manager for one or more resources (hardware or software), for e.g., printers, disk drives, databases etc., whereas clients are the users of server's resources. Thus in a client server model, all shared resources are held and managed by server processes. Client processes issue requests to the servers whenever they need to access one of their resources. If the request is valid, then the server performs the requested actions and sends a reply to the client process.

The client server communication model is oriented towards service provision. An exchange generally consists of:

- transmission of a request from a client process to a server process
- execution of the request by the server
- transmission of a reply to the client

This exchange is depicted in figure B-1

![Client Server Exchange Diagram](image)

This pattern of communication involves transmission of two messages and a specific form of synchronisation of the client and the server. The server process must become aware of the request message sent in step 1 as soon as it arrives. The activity issuing the request in the client process must be blocked or suspended causing a wait until the reply has been received in step 3.

A single server, if implemented as a single process, may take substantial time to process a request. Clients may encounter significant waits while earlier requests are satisfied. If the resource is of the type that can be used by a number of clients
at a time, significantly improved responsiveness can be obtained by providing multiple parallel server processes, any of which can satisfy client requests.

However implementing idle parallel processes may be wasteful. Thus the number of processes chosen is a trade-off between responsiveness and waste of resources. Where the overhead of creating and terminating a server process is negligible relative to the overall processing of a single request, then it may be reasonable to have a multiplexer process waiting to accept requests. When a request is received by a multiplexer process, it can create a server process to handle the request.

As context switching can be quite expensive on the server's resources, servers may therefore be best implemented as threads within a single process. As threads share memory and other resources, there is less overhead in switching from one thread to another than in switching from one process to another.

B2.0 The Three Tier Model

Computing applications can be considered to consist of three types of components. These include the user interface components, the functional processing components and the data access components. The user interface components accept requests from the users and display the results of analysis and processing. The functional processing components perform the necessary processing operations to solve specific problems. The data access components access the data stored in the disk drives or furnished by external processes.

A variety of above mentioned components may be combined in a number of different ways to produce different configurations of client server computing. Thus the computing applications may be termed as the three tier model of client server computing.

Martin & Leben list four fundamental configurations that are possible with the three tier model [MarLeb95]. These may be listed as

- Single computing system model
- User interface distribution model
- Data access distribution model
- Function distribution model

In the single computing model, all the components operate on the same machine (Figure B-2). The user interface distribution model represents a commonly used
form of client server computing. In this model, the application component that implements the user interface executes on one computing environment and the components that perform the function and data processing operate on another computing environment (Figure B-3). This model is generally used when the clients are expected to be machines which are unable to perform complex function processing on extensive data retrieved or acquired from other computing elements.

Figure B-2: Single Computing System Model

Figure B-3: User Interface Distribution Model

The data access distribution model is another widely used form of client server computing. Here the application component that performs data access processing operates on a different computer from application components that perform function processing and user interface processing (Figure B-4).

In the function distribution model, the actual application functions are distributed among a number of different processors. The function processing components of the application might be distributed among one or more client components and one or more server components, each of which can be running on a different computing system (Figure B-5).
Figure B-4 : Data Access Distribution Model

Function distribution can be employed in conjunction with distributing the user interface and data access processing to create powerful applications having any number of components.

Figure B-5 : Function Distribution Model

**B3.0 Communication Mechanisms**

Communication within a client server environment is largely accomplished either by the message passing or by the remote procedure call mechanisms.

Message passing mechanisms consist of a direct exchange of units of data between the client and server components that are running on different computing systems. In its simplest form, a client passes a message to the server, requesting the execution of a process. The message consists of the type of process to be executed by the server as well as the parameters required by the process to execute. Once
the server has carried out the required processing, it sends a set of messages to the client that forms the results of the operation conducted at the server (Figure B-6).

Remote procedure call (RPC) mechanism extends the familiar procedure call programming paradigm from the local computing system environment to a distributed environment. With an RPC facility, the calling procedure and the called procedure can execute on different computing systems in the network (Figure B-7). RPC mechanism attempts to hide from the application programmer the fact that distribution is taking place.

RPC mechanisms are generally built on top of message passing mechanisms and implement a simple request - response protocol. The calling procedure makes a request for the execution of the called procedure and may pass a set of parameters to the called procedure. The called procedure then executes and may pass a set of results back to the calling procedure. The process of formatting and communicating messages between the called and the calling procedure is known as **marshalling**.

RPC has its inherent disadvantages. These include an extended overhead which may be 4 orders of magnitude higher than that of the local call as well as no possibility of sharing memory between the calling and the called procedures.
Use of pointers may also be inconvenient. Finally means of translating references between heterogeneous processing architectures must also be provided.

RPC is usually implemented with connectionless (datagram) protocols such as UDP. Failure of networking devices may therefore inhibit messages to be communicated between the client and the server. An RPC mechanism typically includes a timer mechanism that retries calls or replies which have not been properly acknowledged. There is a possibility that a call may be executed more than one times. This situation is generally dealt either by having the called functions to be idempotent (where multiple calls have the same effect as a single call) or by guaranteeing that any given function is called at most once.

Moreover message passing protocols used in networks generally provide limited length datagrams. These message sizes may not be regarded as adequate for use in transparent RPC systems as the arguments or the results of the procedures may be of sizes larger than these. This problem is generally solved by designing a protocol on top of the message passing operations for passing multipacket request and reply messages [CouDK94]. Thus the requests and replies that do not fit within a single datagram are transmitted as multipackets where the message is made of a sequence of datagrams.

B4.0 Building Remote Procedure Calling (RPC) Clients and Servers

Software supporting RPC provides for
- integrating RPC mechanisms with client and server programs in conventional programming languages
- transmitting and receiving request and reply messages
- locating an appropriate server for a particular service.

An RPC system provides a means for building a complete client program by providing an stub procedure to stand in for each remote procedure that is called by the client program. This client stub procedure converts the local procedure call on the client to a remote procedure call to the server. The types of arguments and results in the client stub must conform to those expected by the remote procedure. This is accomplished by using a common interface definition (Figure B-8).

The task of a client stub procedure is to marshal the arguments and to pack them with procedure identifiers into a message, send the message to the server and then
wait for the reply. Upon receiving the reply, the client stub procedure un-marshals the results and returns the results to the local procedure.

An RPC server system will have a \textit{dispatcher} and a set of \textit{server stub procedures}. The despatcher uses the procedure identifiers in the request message to select one of the server stub procedures and pass on the arguments. The task of the server stub procedure is to un-marshall the arguments and call the appropriate service procedure. When the service procedure ends, the server stub procedure marshals the output arguments or return values (or error output in case of a failure) into a reply message.

Interface definitions are written in a suitable interface definition language and are compiled by an interface compiler. Interface compilers perform the following tasks

- Generate a client stub procedure to correspond to each procedure signature in the interface. The stub procedures will be compiled and linked with the client program.
- Generate a server stub procedure to correspond to each procedure signature in the interface. The dispatcher and the server stub procedures will be compiled and linked with the server program.
- Use the signatures of the procedures in the interface which define the arguments and the result types to generate appropriate marshalling and un-marshalling operations in each stub procedures.
• Generate procedure headings for each procedure in the service from the interface definition. The programmer of the service supplies the bodies of these procedures.

The use of a common interface definition while generating the stub procedures for the client program and the heading for the procedures in the server programs ensures that the argument types and results used by the clients conform to those defined in a server.

The task of the communications module in both the client and the server programs is to deal with communications between them. Generally this is accomplished by a form of a request reply communication. This module is generally provided in forms suitable for linking with the client and the server programs.

Location of an appropriate server for a particular service is accomplished by means of binding. An interface definition specifies a textual service name for use by clients and servers to refer to a service. However client request messages must be addressed to a server port.

Binding specifies a mapping from a name to a particular object, usually identified by a communications identifier. The binding of a service name to the communications identifier specifying the server port is evaluated each time a client program is run. Binding is essential when server port identifiers include host addresses, as it avoids the need to compile server port identifiers into client programs. If a client program does include the host address of a server, it will need to be changed (even re-compiled in some cases) whenever the server is relocated.

A binder is a separate service that maintains a table containing mappings from server names to server ports. A binder is intended to be used by servers to make their port identifiers known to potential clients. When a server process starts executing, it sends a message to the binder requesting it to register its service name and port. If the server process terminates, it should send a message to the binder requesting it to withdraw its entry from the mapping.

When the client process starts, it sends a message to the binder requesting it to look up the identifier of the server port of a named service. The client program
sends all its request messages to this server port until the server fails to reply, at which point the client may contact the binder and attempt to get a new binding.

**B5.0 Remote Method Invocation (RMI) in Java**

Java Version 1.1 provides a Remote Method Invocation (RMI) feature. This enables a program operating on a client computer to make method calls on an object located on a remote server machine. Thus a Java programmer can distribute computing across a networking environment.

RMI defines a set of remote interfaces that can be used to create remote objects. A client can invoke the methods of a remote object with the same syntax that it uses to invoke methods on the local object. The RMI API provides classes and methods that handle all of the underlying communications and parameter referencing requirements of accessing remote objects. RMI also handles the serialisation of objects that are passed as arguments to the methods of the remote objects.

The RMI consists of the **stub/skeleton, remote reference** and the **transport** layers (Figure B-9).

![Java RMI Architecture](image)

**Figure B-9 : Java RMI Architecture**

When a client invokes a remote method, the request starts at the top of the **stub** on the client side. The client references the stub as a proxy for the object on the remote machine. Stubs define all the interfaces that the remote object implementation supports. The stub is referenced as any other local object by the program running on a client machine. It looks like a local object on the client side. It also maintains a connection to the server side object. The **Remote Reference Layer** on the client side returns a marshal stream to the stub. The marshal system
Client Server Computing

is used by the Remote Reference Layer (RRL) to communicate to the RRL on the server side. The stub serialises parameter data, passing serialised data to the marshal stream.

After the remote method has been executed, the RRL passes any serialised return values back to the stub, which is responsible for de-serialising.

The skeleton is the server side construct that interfaces with the server side RRL. The skeleton receives method invocation requests from the client side RRL. The server side RRL must un-marshalling any arguments that are sent to a remote method. The skeleton then makes a call to the actual object implementation on the server side. The skeleton is also responsible for receiving any return values from the remote object and marshalling them onto the marshal stream.

The Remote Reference Layer maintains an independent reference protocol that is not specific to any stub or skeleton model. This allows for changing the RRL, if desired, without affecting the other two layer. The RRL deals with the lower level transport interface and is responsible for providing a stream to the stub and skeleton layers.

The transport layer is responsible for creating and maintaining connections between the client and the server. The transport layer sets up the connections, maintains the existing connections and handles the remote objects existing in its address space.

The transport layer, upon receiving a request from the client side RRL, locates the RMI server for the remote object that is being requested. The transport layer then establishes a socket connection to the server. The established connection is then passed to the client side RRL and a reference to the remote object is added to an internal table. At this stage client is considered to be connected to the server.

The transport layer disconnects the client and the server if there is no activity on the connection for a considerable period of time. This time-out period is 10 minutes.

B6.0 Implementing Client Server Systems in Java

Steps required in implementing a client server system in Java are similar to those required in other RPC based client server systems. These include
- defining interface for the remote classes
- creating and compiling implementation classes for the remote classes
- creating stub and skeleton classes
- creating and compiling a server application
- starting the RMI registry and the server application
- creating and compiling client programs to access remote objects

The interface classes extend the `java.rmi.Remote` class.

Implementation classes for the remote objects extend `UnicastRemoteObject` class. This defines the remote object as a unicast object, which means that it can only handle one client reference at one time. If more than one clients attempt to access the remote object, they will be queued up and will receive the object reference only when the current reference is given up.

Once the implementation classes have been created and compiled, stub and skeleton classes need to be generated. The stub class is used by the client code to communicate with the server skeleton code. These classes are generated with the help of `rmic` command. This command automatically creates stub and skeleton code from the interface and implementation class definitions.

The server needs to publish an object instance by binding a specified name to the instance and registering that name with the *RMI Registry*. An instance can be bound either with the help of the `bind` or the `rebind` method calls in the *Naming* class. These methods are static and require, as parameters, a name to reference the object as well as the actual remote object that is to be bound to the name. The name is to be supplied as a URL-like string, i.e. in the format `protocol://host:port/bindingName`. Here the *protocol* should be *rmi*, *host* is the name of the RMI server, *port* is the port number on which the server should listen for requests and *bindingName* is the exact name that should be used by the client when requesting access to the object.

`bind` and `rebind` only differ in the behaviour when the name being bound has already been bound to an object. In this case `bind` throws `AlreadyBoundException` whereas `rebind` will discard the old binding and enforce the new one.
Naming lookup service is provided by the RMI Registry. It is an independent program that must be running before the server application is invoked. This program can be invoked as a background process by simply typing `rmiregistry &` at the operating system prompt. Once an object has been passed to the Registry, a client may request that the RMI Registry provide a reference to the remote object.

The client uses the `lookup` method in the Naming class to get a handle to the required remote object. The `lookup` method requires the server name as a URL-like string to be passed as a parameter. This method communicates with the server and return a handle to the requested remote object.
Appendix C

Client Server Databases

C1.0 Introduction
Client server database technology aims at managing the data and information as a resource that may be required by multiple clients for different applications. In the simplest client server database systems, the application programs are executed on the client machines. They access the required data from the servers over a network. This data is accessed via one or more complementary layers of database software that reside on the clients as well as the server. Only the local database software is visible to the client application and it appears as if the data is stored locally when in fact it may be physically located at a great distance away.

In a more complicated scenario, the database may itself be divided up and stored in a number of remote machines. Again the local application program uses the client database software to request access to the data as if it were all stored locally. Server database software running on a number of remote machines co-operate to access the required data on behalf of the local client database software and the application program (Figure C-2).

There are a number of advantages that motivate the development of distributed databases. These range from organisational and economic reasons (Information systems designed to match the organisational structure) to operational considerations (performance enhancements, reliability and availability). Moreover distributed database systems offer a potential for smooth incremental growth [CerPeI85]. Thus with the addition of a new organisational function, its associated
Databases can be implemented over suitable servers connected to the organisational network.

Development of high performance clients has added a new dimension to the research in client server databases. Efficient means of processing queries are being investigated that will improve the overall performance of the client server databases. This is being accomplished by adaptively utilising the computing resources at the clients and the servers in order to process the queries. This includes caching the results of previous queries at the client and re-using them to answer subsequent queries, partially or completely. This reduces the requirement to access the server, thereby reducing the overall response time as well as increasing the server availability. System scalability is also improved significantly. Moreover utilisation of client resources adds to system predictability. If the required data can be obtained locally, access time (as well as the performance) is independent of the activity of other users [Fra93].

Client server databases can be categorised in two ways. One categorisation is based on the location of the application processing components, database software components and database. These systems may also be categorised on the basis of caching of data in the client machines and utilisation of client resources to process the data and provide results.

**C2.0 Database Categorisation In Relation to Component Location**

Martin & Leben have categorised database systems into five architectural models, based on the location of different components that make up these systems [MarLeb95]. These models are described as follows.
C2.1 Centralised Database Model

In the centralised model, application processing components, the database software and the database itself, all reside on the same processor (Figure C-3).

Figure C-3: Centralised Database Model

Much of the mainstream information processing performed by many organisations still conforms to the centralised model. These information systems may provide terminal operators at widely distributed locations with fast access to a central pool of data. However, in many such systems, all three components of the database application execute on the same mainframe. Thus this configuration also conforms to the centralised model.

C2.2 File Server Database Model

In this model, the application components and the database software reside on one computing system and the physical files making up the database reside on some other computing system (Figure C-4).

Figure C-4: File Server Database Model
Such a configuration is often used in a local area network environments where one or two computing systems play the role of file servers that store data files. Other computing systems have access to these files. Networking software used makes the application software and the database software running on an end user system think that a file or database stored on a file server is actually attached to the end user machine.

The file server model is similar to the centralised model. The files implementing the database reside on a different machine from the application components and the database software. The application components and the database software may be the same as those designed to operate in a centralised environment.

This environment can be more complex than the centralised model as the networking software may implement concurrency mechanisms that permit more than one end users to access the same database without interfering with one another.

**C2.3 Database Extract Processing**

In this model, the user makes a connection with the remote computing system where the desired data is located. The user then interacts directly with the software running on the remote machine and formulates a request to extract the required data from the remote database. The desired data is then transferred from the remote computer to the local computer where it is processed using the local database software (Figure C-5).

![Diagram of Database Extract Processing Model](image)

**Figure C-5**: Database Extract Processing Model
In such systems, the user must be aware of the location of data as well as the means to communicate with the remote computer and access the desired data. Complementary application software must be available on both computing systems to handle database access and the transfer of data between the two systems. However the database software on these machines need not be aware that remote database processing is taking place as the user interacts with them separately.

C2.4 Client Server Database Model

In a true client server database model (Figure C-6), the database resides on a machine other than those that run the application components. The database software is however split between the client system (that runs the application programs) and the server system (that operates the database).

Here the application processing components on the client system make requests to the local database software. The local database software component in the client machine then communicates with complementary database software running on the server. The server database software makes requests for accesses to the database and passes the results back to the client machine.

This model may appear similar to the file server database model, but presents significant advantages as compared to it. Network traffic is reduced significantly for the same database operations if implemented using a client server model. In a file server model, the entire data related to a query is transferred to the client where the query is processed. On the contrary, only the results of the query are transferred by the server to the client in the client server model.
C2.5 Distributed Database Model

Ceri & Pelagatti list two properties of distributed databases [CerPeI85],

- **Distribution**, i.e. the fact that the data are not resident at the same site (processor) so that it can be distinguished from a single centralised database.

- **Logical correlation**, i.e. the fact that the data have some properties which tie them together so that a distributed database can be distinguished from a set of local databases or files which are resident at different sites of a computer network.

Thus a distributed database is said to exist when a logically integrated database is physically distributed over several distinct linked computing facilities [Dav78].

![Distributed Database Model](image)

**Figure C-7 : Distributed Database Model**

One of the major issues in distributing databases is the optimal distribution of components. In distributed database systems, data as well as functions can be distributed [Bra82].

There are four main techniques of distributing the database. Data may not be physically distributed at all. The database is then itself centralised but the access to the database is distributed. Such systems although simple to develop and maintain, present a single point of failure.
Alternatively complete copies of database are placed at different nodes. This can result in major improvement in the reliability of the systems. Moreover the communications costs as well as the response times are reduced significantly. Serious complications may, however, arise for updates as all the copies of database must be synchronised. Different techniques exist to synchronise the replicas with the main database. The update may be made in the appropriate replicas and then the change is propagated across to the main database. Alternatively the main database may be updated along with the appropriate replicas.

Replication must be distinguished from extraction. In extraction, the database copy is intended to be used on a read-only basis. Data element values in an extract are not intended to be updated [MarLeb95].

The database may be partitioned, i.e. it is divided into disjoint sets with each part of it being placed at different nodes. Since there is no overlap, there is only one copy of data. This means that there is no consistency problem. Synchronisation only becomes a problem for some compound requests that require data from several nodes. If data is partitioned appropriately and certain types of requests are not allowed, the synchronisation problem can be completely avoided. Partitioning is practical only if the data exhibits a high locality of reference i.e., most of the requests for the particular part of the data only come from a single node at which data can be stored. In other cases where other nodes are equally likely to use the data, partitioning would simply increase the traffic over the network.

Finally a hybrid approach may be applied wherein different databases or parts of these databases are placed at different nodes using a combination of the above mentioned techniques.

**C3.0 Database Categorisation on the Basis of Client Data Caching**

Ryan & Smith categorise client server database systems according to the amount of database processing and local caching done at the clients. They suggest that client caching and processing is based on the granularity of the database accesses [RyaSmi95].

*Coarse grained* applications are characterised by transactions that access large volumes of data, for e.g., decision support systems etc. These applications have
few large interactions with the databases, but they might benefit from the parallel execution of queries on the servers. If complex queries cannot be processed at the servers, potentially useful data is communicated to the client for processing. However with the advent of powerful data manipulation languages and database systems, the ability of database servers to process a wide range of complex queries is improving. For such systems, clients are generally concerned with presentation services. Clients and servers are connected only by streams of queries and results.

On the other end of spectrum, fine grained applications are typically navigational, accessing small portions of linked records (for example, CAD, CASE applications). It is advantageous for a fine grained application to request a large data transfer from the server to a local cache. In such applications, there are frequently large numbers of references internal to a complex object and very few to other objects. This attempts to minimise the cost of crossing between the database and application, at the cost of maintaining the cache coherency. Since fine grained operations often have a high locality of reference, this approach can give good performance. Performance deteriorates significantly for poor locality of reference.

Based on the exploitation of the client resources and the location of query execution, client server database systems are generally categorised as query shipping or data shipping systems [KosFra95], [FraJK96]. Data shipping specifies that all operators of a query should be executed at the client machine at which the query is submitted. Object oriented database systems are typically based on data shipping where the required data is faulted to the client to be processed as well as cached there. Data shipping approach has the benefits of exploiting the resources of powerful client machines and reducing communication in the presence of locality or large query results. This also improves scalability as users and their desktop resources are added to the system [FraJK96].

Query shipping is a widely used term in the context of a client server architecture with one server machine where queries are completely evaluated. Relational systems and their descendants are typically based on query shipping. Query shipping reduces communication costs for high selectivity queries, posses the ability to exploit server resources when they are plentiful and the ability to tolerate low performance client machines [FraJK96].
It has been stressed that neither query shipping nor data shipping is the best policy for query processing in all situations [KosFra95], [FraJK96]. As a result, a system that supports only one of these policies is likely to have sub-optimal performance under certain workloads and/or system conditions. A hybrid approach has been suggested that can outperform both the pure policies. Hybrid shipping combines the approaches of data and query shipping. It allows the most efficient execution of a query, but is also the most difficult policy to optimise. Moreover, hybrid shipping does not preclude a relation from being shipped from a client to a server for processing if, for example, the relation is cached in the client's main memory and further processing is more efficient at the server [KosFra95].

Hybrid shipping has been shown to [FraJK96]
- execute query operators at clients if clients' resources are at least at parity with server resources.
- execute query operators on servers when parallel and/or plentiful server resources are available.
- exploit caching to reduce communication costs in some cases but does not use cached copies of data if the relations used in the query are co-located at a server.
- can exploit caching to reduce interaction with heavily loaded servers but ignores cached copies of data if server resources are plentiful and the use of cached data at the client would increase the response time of a query by causing contention of client's disks.

Hybrid shipping, however, results in complex query optimisations where pre-compiled query plans can be sensitive to changes in system state and location.
Appendix D

Object Wrapper for Primitive Network Performance Data

D1.0 Introduction
Object technology allows developing applications that can effectively model the organisation and structure of real world environments. Objects can concisely describe the semantics of the application and facilitate reuse. Thus objects allow organisation of data according to the needs of the applications.

An increasing number of object oriented applications are being developed. Data used by a large number of these applications reside in relational, hierarchical and network databases and legacy file systems. These data are shared between the new object oriented applications and the existing non-object oriented applications. It may also not be feasible for organisations attempting to migrate to object oriented paradigm to convert their existing databases to object oriented databases for the following reasons [KapPR94], [KelTur95],

- migration from existing databases to object oriented databases may be extremely expensive.
- if existing databases are converted to object oriented databases, all existing applications have to be re-implemented using object technology.
- the object schemata are structured according to the needs of a set of applications. Thus new applications or modifications to existing object oriented applications may not conform entirely to the original requirements.

Thus such environments present a mismatch between the programming model (objects) and the manner in which existing data are stored (for e.g. relational tables). An approach employed commonly to reduce this mismatch is to develop object wrappers that access non object data and return that data to an application in object format.

This appendix briefly explains an object wrapper that has been developed to access the primitive network performance data. The network performance data collected by the test stations is downloaded to a control and data processing workstation periodically. This data is integrated into a database. The database is partitioned such that one partition consists of monitoring data for a particular month of a year.
Each partition consists of two tables for each test conducted by the test stations. A test is conducted by transmitting test packets of a specific size on a specific route or link. Thus one test differs from another either with respect to the size of the test packet used in the test or the route being monitored. One table contains data regarding the test packets transmitted by the transmitting test station during a particular test. The other table contains data regarding the test packets received by the receiving test station for that particular test.

All queries regarding primitive network performance data are posted to the object wrapper. The object wrapper accesses the appropriate tables and retrieves the required records. These records are returned as appropriate objects to the client applications.

**D2.0 Database Structure for the Primitive Performance Data**

The data regarding the test packets transmitted by the transmitting test stations and the test packets received by the receiving test stations are maintained in similar tables. The structure of the table maintaining the data regarding the test packets transmitted by the transmitting test stations is given below

\[
\text{TX\_TABLE} = (\text{TX\_TIME, TX\_USECS, GAP, TEST\_ID, SIZE, PACKET\_ID})
\]

TX\_TIME represents the time of transmission of a test packet in milliseconds since 1 January 1984. TX\_USECS represents the time in microseconds from TX\_TIME when the test packet is transmitted from the transmitting test station. GAP represents the time in microseconds between the start of transmission of the test packet from the transmitting test station and the end of transmission of the specific test packet.

TEST\_ID represents the test identifier. It is a unique identifier for a particular type of monitor experiment. It represents the route, network link or the device being monitored and the size of the test packets used etc. This may also be termed as the configuration data for a particular monitoring experiment [Sho91]. SIZE represents the size of the test packet in bytes. PACKET\_ID represents the packet identifier. It is a number assigned to each packet when it is composed at the transmitting test station.
The structure of the table maintaining the data regarding the test packets received by the receiving test stations is given below

\[
\text{RX_TABLE} = (\text{TX_TIME}, \text{TX_USECS}, \text{DELAY}, \text{TEST_ID}, \text{SIZE}, \text{PACKET_ID})
\]

The only field that is different in the table of received test packets from the table of the transmitted test packets is that of DELAY. This represents the delay in microseconds that is experienced by a test packet as it is communicated from the transmitting test station to the receiving test station.

All fields in both the tables store data as integer numbers, each being 4 bytes long.

The primitive monitoring data for each test conducted by the monitoring system is maintained in a separate pair of the above mentioned tables. This data is further partitioned into tables in different directories, one for each month of the year (Figure D-1).

![Figure D-1: Partitioning Primitive Monitoring Data](image)

The test packets generated by the transmitting test stations are naturally ordered chronologically and are logged in the same order. This eliminates the need for a sort operation in any of the tables maintaining the data regarding the test packets transmitted by the test stations. However the test packets may not be received by the receiving test stations in the order in which they have been transmitted by the transmitting test station. This necessitates a sort operation on the tables maintaining the data regarding the test packets received by the receiving test stations.
As the transmit time of each packet transmitted by the transmitting test station is different, it may be used, along with the packet id, to uniquely identify each record representing a packet. The packet id may reset prematurely under certain circumstances, for e.g., test station reboot. Thus the transmit time of the packets may be used as keys for tables containing data for the transmitted as well as received test packets.

D3.0 Objects for Test Packet Data and Query Specification

In the simplest form, the operation of a database object may be explained as shown in figure D-2. The client object or application passes a query specification object to the appropriate method of the database object. The database object returns an object containing the results for the query specification object. the database object may also throw various exception at the client object. These exceptions indicate the occurrence of abnormal events, for e.g. no more data for the query specification, database failure etc.

The data retrieved from the primitive monitoring database by the object wrappers is returned to the client application as objects representing test packets, one for each record in the transmitted or received data table being read. The object wrapper described in this appendix returns an object of the TestPacket class for each record that is retrieved form a database table as a result of a particular query specification.
The TestPacket class extends a Packet superclass. An object of the Packet class is used to represent the characteristics of a generic data packet. An object of the Packet class contains the following information

- transmit time of the packet in milli-seconds since 1 January 1984
- time in microseconds from the transmit time when the data packet is transmitted
- packet identifier
- packet size in bytes
- test identifier which represents the source, the destination and the size of the packet as a coded integer value

Objects of the Packet class use an object of the LuTime class to manipulate the time and date information.

An object of the TestPacket class represents a specific type of test packet used for intrusively monitoring the data communication networks. In addition to the above mentioned information, an object of the TestPacket class representing a received test packet also contains the delay a received test packet experiences as it is communicated. Similarly an object of the TestPacket class representing a transmitted test packet contains the time difference between the start and end of transmission of the test packet from the transmitting test station.

Objects representing query specifications to the primitive database are instantiations of the QuerySpecification class. This class extends an abstract StoreQuery class. An object of the QuerySpecification class, when instantiated, requires the start and end times for the query specification and an integer representing the test identifier for which the data is to be retrieved. The start and end times for constructing the QuerySpecification object are objects of the LuTime class and represent the transmit time of the first and the last test packets to be included in the result of the query. The constructor of the QuerySpecification class throws an IllegalQueryException if the start time is greater than the end time.

D4.0 Object Wrapper for The Database Tables
An object of the RawDB class is used as an object wrapper for a table of the primitive monitoring database. Objects of this class provide read only access to the specified database tables.
An object of the **RawDB** class is instantiated by passing the following parameters to the constructor method:

- string representing the type of database to be accessed for a particular test id: *(**tx** to access the table of transmitted test packets, **rx** to access the table of received test packets)*
- string representing the database directory
- an object of the **QuerySpecification** class specifying a query to the database

From the parameters passed to it, the constructor uses a method to construct the complete name of the file containing the required data (including its path). For example, if the database directory is `/lee/hsnl/dbdir`, the start time in the query specification is `1996/11/3:6:0:0` and the test identifier is `5101`, the name of the file containing the data regarding the transmitted test packets is `/lee/hsnl/dbdir/1996/11/5101.tx`. Similarly, the name of the file containing the data regarding the received test packets is `/lee/hsnl/dbdir/1996/11/5101.rx`.

The constructor attempts to open the database file. The constructor throws an **IOException** if an error occurs during this operation. After opening the specified database file, the pointer of the database file is moved to the first record in the file that forms the result for the query specification. As all the database tables are sorted in chronological order, binary search algorithm is employed to expeditiously move the database pointer to the desired position.

Each trigger to the `get_a_test_packet()` method of the object wrapper returns an object of the **TestPacket** class for the record in the specific file that is currently pointed at by the file pointer. If a trigger to the `get_a_test_packet()` forces the file pointer beyond the end of file, the object wrapper closes the file currently open and opens the database file containing the data for the same test identifier and data type (transmitted or received packets data) for the next month. After opening the new database file, the `get_a_test_packet()` method throws **EOFException**. In case an error occurs while attempting to access the database file, the `get_a_test_packet()` method throws **IOException**.

Client applications can close the database files by triggering the `closeDB()` method of the wrapper. This method is also called by the destructor (`finalise()` method) for the **RawDB** class.
D5.0 Query Manager
A client application may request data for a number of different test conducted by the test stations for the time period specified in a particular query specification. For each test identifier and the data type i.e., transmitted or received test packet data, a separate object of the RawDB class is instantiated. A query manager object of the QueryResponse class can be used to retrieve the data objects from each of these object wrappers.

An object of the QueryResponse class is instantiated by passing an object of the QuerySpecification class to it. Object wrappers of the RawDB class for all the database tables from which data needs to be retrieved for the specified period have to be instantiated separately. Each of these objects of the RawDB class is passed as a parameter to the getNext() method. This method returns an object of the TestPacket class representing the data record being currently pointed at in the database table handled by the particular object wrapper.

If the TX_TIME field of the record retrieved from the database table is greater than the end time specified for the query, the getNext() method of the query manager throws a QueryEndException at the client object. This indicates that no more data for the period specified in the query exist in that particular database table.

D6.0 Enhancing Wrapper Performance
Following methods may be employed to enhance the performance of the object wrapper

D6.1 Accessing Specific Data Fields
The object wrapper, in its present state, access and retrieves complete records from the specified database tables and returns objects of the TestPacket class corresponding to the records retrieved. The object wrapper can be modified so as to retrieve data from only the specific fields in the database tables. For example, if a client application only requires delay experienced by each test packet for a particular query specification, only the TX_TIME and DELAY fields may be accessed. Similarly data only in the TX_TIME and PACKET_ID fields is required to calculate the packet loss and duplication information.
This reduction in the amount of data retrieved from the database tables can significantly enhance the performance of the object wrapper for information applications that do not need all the data in the records of the database tables.

D6.2 Indexing the Database

The volume of the primitive monitoring data justifies the use of indexes for the tables to accelerate the retrieval process. Indexes are maintained as tables of key values and record locations for the records corresponding to the key value. As mentioned previously, the transmit time of the packets are used as keys to uniquely identify records of packet details.

Due to the significantly large amount of data in the database, dense indexing [Dat95], [RyaSmi95] would also result in large index tables. However as data is retrieved from the database as datasets or combination of contiguous datasets, a sparse index can be conveniently used to index the database. The index tables can maintain the indexes to the first record of each dataset that is downloaded from the test stations and appended into the database.

Upon instantiation, the object wrapper may search the index to find the location of the database record that is above and is closest to the database record required by the query. The pointer of the database table may then be positioned to the desired location by using the binary search algorithm on the records between the current location of the file pointer and the end of the database file. The index tables are also sorted in chronological order. Thus binary search algorithm may also be applied on the index tables.

Figure D-3 : Dense Index vs. Sparse Index
The size of the index tables is expected to be much smaller as compared to the size of the corresponding database table. Combination of appropriate indexing and search methods are therefore expected to enhance the performance of the object wrapper significantly.

**D7.0 Summary**

This appendix describes an object oriented wrapper and associated classes used to access the primitive database tables. The example presented here briefly describes their use in a data processing application (Figure D-4).

![Diagram](image)

**Figure D-4 : Example Primitive Data Processing System**

The data processing object instantiates two objects of the **RawDB** class and an object of the **QueryResponse** class on receiving an object of the **QuerySpecification** class from the client object. One object of the **RawDB** class provides a handle to the transmitted packet table whereas the other provides a handle to the received packet table. If an error occurs during the instantiation process, an **Exception** is thrown by the data processing object at the client object.

The client object can retrieve the data from the respective database tables by triggering appropriate method of the data processing object. This object in turns triggers the `getNext()` method of the object of the **QueryResponse** class. Appropriate database wrapper object is passed as a parameter to this `getNext()`
method. This method triggers the `get_a_test_packet()` method of that database wrapper object to retrieve the record being pointed at in the database table. This wrapper instantiates an object of the `TestPacket` class for the retrieved record and returns it to the object of the `QueryResponse` class. The `getNext()` method returns this object of the `TestPacket` class, which is ultimately returned by the data processing object to the client object. Once the TX_TIME of the record retrieved from the database table exceeds the end time of the query specification, the object of the `QueryResponse` class throws `QueryEndException` at the client object.
Appendix E

High Level Network Performance Information: Incidents and Effects

E1.0 Introduction to Network Incidents and Effects

The primitive network performance data passes through several processing and analysis stages before suitable performance information can be derived. The network performance analysis operation is shown in figure E-1. The primitive monitoring data is appropriately summarised. The summary derivation process is generally the most computationally intensive stage. Several issues need to be addressed at this stage. These include the determination of suitable types of summaries that need to be derived to assist the analysis process. Moreover these summaries need to be derived at an appropriate level of granularity.

These summaries are analysed with the primitive data and the network configuration and status information. This allows to determine different network events or network incidents that occurred over the network links and devices during the analysis period. These incidents include events like excessive packet losses, significant delay variations, loss of service etc.

Recurring and correlated incidents are categorised into network effects. These usually indicate the characteristics of the network elements and host applications as well as various faults such as improper system configuration. For example, regular variations of delay values may indicate a certain characteristic of the
network switches which may delay some packets significantly at regular intervals. Alternately, some host applications may transmit unnecessarily large amount of data over these links at regular intervals. This may delay the network traffic significantly.

Network performance analysis is an iterative process. The analyst may want to investigate the incidents and effects by deriving different summaries at various levels of granularity. The summary derivation process therefore has to employ techniques that can efficiently process primitive data and appropriate summaries to generate the required summaries.

Network incidents and effects are documented as Incident Reports and Effect Reports. This information regarding various detected trends and events is usually maintained in a subjective and narrative manner. These reports need to be maintained on a historical basis.

Each Incident Report is identified by a unique Incident Report number. If an Incident Report describes any change in a previously detected incident or event, it updates the Incident Reports raised previously to describe those events. In addition to the textual description of the incident, graphic representation of the data related to the performance summaries may also be included. A sample Incident Report is shown in figure E-2.

Incident Report - T/97/53
Instigator: Omar Bashir
Date: 17 November 1997
Number of sheets attached: 01

Description:
The average delay value for the test packets transmitted from Station A01 to Station B01 increased significantly on 2 November 1997 at 0000 hours. This average delay value reduced to normal average delay at 0000 hours on 5 November 1997.

Sheet 1
This sheet shows the above mentioned effect

Average Delay : Test ID 1012

Figure E-2 : An Incident Report
This appendix explains a simple database application that is used to manage the information in Incident Reports raised after analysing the network performance data and summaries.

**E2.0 Strategies for Management of Incident Reports**

Incident Reports previously generated and maintained in an information system (automated or manual) may be retrieved in response to queries specifying:

- Incident Report numbers. In this case the user may require the ability to navigate through all those Incident Reports sequentially that describe the effect being studied.
- Time periods. The user may wish to know all the Incident Reports raised during a specified time period.
- Performance tests. Examination of the historical behaviour of specific network elements may be the objective here.

A text retrieval system may be used to retrieve documents related to specific Incident Reports in response to user queries. Text retrieval systems focus on the storage and retrieval of text rather than numeric, tabular or graphical data. Text retrieval systems are designed to provide a range of access points to a database of relatively unstructured information (known as free text) [Row87].

![Text Retrieval System for Incident Reports](image)
Queries are specified as terms and phrases, documents related to which the user wishes to retrieve. The retrieval system may list the Incident Report number (and possibly a short summary) of each Incident Report that contains the specified terms or phrases. From this list, the user may select and view relevant Incident Reports (Figure E-3).

The Incident Reports may be organised as hypertext documents. Hypertext is a collection of information joined by pre-established associative links. Readers can move from one document to another via the links. When different media such as video, sound and animations are included with the text in the branching structure, the system is known as hypermedia.

Hypertext and hypermedia can be regarded as databases containing large quantities of information which can be accessed using various specially devised navigation tools. These enable the user to browse and search the information. Nodes are the content structures of a hypermedia/hypertext system and links are the relationship structures [Pre94]. As the hypertext databases increase in size, navigation or browsing has to be augmented with information retrieval through queries.

The Incident Reports in a hypertext system may be linked via Incident Report numbers. The test identifiers and dates may be arranged in a search index or a search facility allowing convenient access to the relevant Incident Reports (Figure E-4).

Figure E-4: Incident Report Management using Hypertext
Incident Reports are relatively structured documents. It is possible to organise the information in these documents such that these may be managed by a suitable database management system, for example a relational or an object oriented database. Here the documents are analysed and the structure of these documents is determined. This allows to structure the database to manage the information in these Incident Reports. It is essential that the information in these documents adheres to this common structure. Thus the user extracts the relevant information from each Incident Report and adds it to the appropriate fields in the database records.

E3.0 A Database of Network Incidents
The following paragraphs describe the structure of the database designed to manage the network incident details. This database has been developed using the Microsoft Access relational database. A prototype front-end application has been developed in Microsoft Visual BASIC 4.0. This front-end allows the manipulation of the incident details in this database.

E3.1 Database Structure
Each Incident Report is identified by a unique Incident Report number. However an incident may be raised due to an event occurring on a number of different routes. Thus there may be more than one test associated with a specific incident.

The network performance incident data may therefore be maintained by two separate tables in a relational database. One table maintains the incident details (minus the associated test identifiers) and the second table contains the test identifiers associated with each Incident Report. The table for incident details contains the records maintained as follows,

\[
\text{INC REP} = (\text{INCIDENT NUMBER} : \text{text}, \text{PREVIOUS INCIDENT} : \text{text}, \\
\quad \text{START DATE} : \text{date/time}, \text{START TIME} : \text{date/time}, \\
\quad \text{END DATE} : \text{date/time}, \text{END TIME} : \text{date/time}, \\
\quad \text{INCIDENT SUMMARY} : \text{memo})
\]

The table for test identifiers associated with the Incident Reports contains records maintained as follows

\[
\text{TEST ID} = (\text{INCIDENT NUMBER} : \text{text}, \text{TEST IDENTIFIER} : \text{text})
\]
Each record in the incident details table describes the start and end time of each incident, the Incident Report number and a short summary of the detected event. Another field contains the Incident Report number of the latest related incident. This allows the user to navigate through the related incidents.

Each record in the other table contains the test identifiers associated with each Incident Report numbers. For each Incident Report number there may be more than one test identifier. However the combination of the Incident Report number and the test identifier in a record is unique in the table.

The system also allows storage and retrieval of one graph for each link monitored by the system. These graphs are stored as bit mapped files. The names of the files are combination of the numeric digits in the Incident Report number and the digits indicating the specific network link in the test identifier. This allows for efficient retrieval of relevant graphs with the incident details.

**E3.2 Front-end Application**

The front-end application for the database described in the previous section provides a suitable user interface and appropriate data manipulation mechanisms. The following paragraphs describe these data manipulation operations.

**E3.2.1 Data Append**

In order to add an Incident Report into the database using this prototype front-end, the incident details and the relevant test identifiers need to be added separately into the appropriate tables. These data are added into the appropriate tables using Incident Data and the Test IDs forms.

The **Incident Data** form allows the user to type the incident details (minus the test identifiers) in the appropriate fields in the form (figure E-5). After the user has typed in these details, the data may be added to the table for incident details by clicking on the **Add** button. A transaction is committed only if all the fields in the form, except the **Updates Incident** field, contain data in them, the Incident Report number is a unique incident identifier and the date and time fields contain valid data. The user may cancel the transaction by clicking on the **Cancel** button. Once a transaction has been committed, the system provides the user with a blank **Incident**
Data form to enter details of a new Incident Report. The user may close this form by clicking on the Cancel button.

Figure E-5: Incident Data Form

Figure E-6 shows a Test IDs form. One form is used to add a particular test identifier of a specific Incident Report. The transaction is committed if the combination of the test and the incident identifiers is unique. The transaction may be committed by clicking on the Add button. Alternately it may be cancelled by clicking on the Cancel button. After each transaction has been committed, the system provides the user with a blank Test IDs form to enter a new combination of the test and incident identifiers. This form may be closed by clicking on the Cancel button.

E3.2.2 Data Update
Data in these tables may be updated with the help of this facility. In order to update incident details in the Incident Report table, the user needs to retrieve the current data by specifying the Incident Report number. The data for that Incident Report is displayed on the Incident Data form (figure E-7). After the user has altered the data on the form, it may be updated in the table by clicking on the Update button. Alternately the user may cancel the transaction by clicking on the Cancel button. The transaction is not committed if any of the fields (except the PREVIOUS_INCIDENT field) is blank, the data in the
INCIDENT_NUMBER field is not unique or the date and time fields contain invalid data and time values.

**Figure E-6 : Test IDs Form**

In order to update the data in the table for the test identifiers associated with the Incident Reports, the user needs to specify the Incident Report
number and the test identifier to be updated. The data to be updated appears in the Test IDs form (Figure E-8). Once the user has corrected the data in the form, the table may be updated by clicking on the Update button. The transaction may be cancelled by clicking on the Cancel button. The transaction is not committed if the combination of the numeric digits in the Incident Report number and the digits in the test identifier representing the link being monitored is not unique or any one of the fields is blank.

![Incident Reports Database](image)

Figure E-8 : Updating Test Identifiers

E3.2.3 Data Retrieval

As mentioned previously, Incident Reports may be retrieved by specifying the time period between which these incidents occurred, the Incident Report number or the test identifier.

The data retrieved by the system is returned in the Query Results form (figure E-9). This form contains the details of an incident and the test identifiers on which the particular event was observed. The user may view the performance summary for a specific link, on which the event was observed, in a graphical format by clicking on the test identifier in the Test Identifiers list box. The system constructs the relevant bit mapped file name from all the numeric digits in the Incident Report number and those digits in the selected test identifier which represent the monitored link. The
bit mapped image in the file is displayed in a picture box in the Performance Graph form (Figure E-10).

**Incident Reports Database**

![Incident Reports Database](image)

*Figure E-9: Incident Data Retrieval*

![Performance Graph](image)

*Figure E-10: Performance Graph*
If the data is retrieved for a specific time period, the system access the table of incident details and returns the data for earliest incident within the specified period. The system then accesses the table for the test identifiers and returns all the test identifiers for the Incident Report number retrieved. These details are displayed to the user on the Query Results form. If the user clicks on the Previous or Next buttons on this form, the system displays the details of the incident that occurred immediately before or after the incident being viewed as long the time of occurrence of that incident falls within the specified period.

It is also possible to view the historical performance of a specific link for a particular test identifier. The system displays the details of the latest incident that occurred for the specified test identifier. The user can view the earlier incidents for that particular test identifier by clicking on the Previous button. From a particular Incident Report being viewed, the user may move to a more recent incident reported for the specified test identifier by clicking on the Next button.

For a specific Incident Report retrieved by the user, it is possible of view all the related Incident Reports in the database. From a specific Incident Report being viewed, the user may view the details of the Incident Report that is updated by this Incident Report (identified in the Updating Incident Number field of the Query Results form) by clicking on the Previous button. Similarly it is possible to view the details of the Incident Report that updates the Incident Report being viewed (i.e. the Incident Report for which Updating Incident Number contains the current Incident Report number) by clicking on the Next button. This facility allows the user to view the overall effect.

The Write As Text button allows the user to prepare a text report of the incidents retrieved by the system. Clicking on this button invokes the Save Results in File form (Figure E-11). The user may save the incident details of the incident being viewed or all the related incidents. If the Current Record radio button is selected, the system saves the details of the incident being viewed in the text file specified by the user. However if the All Matching Records radio button is selected, the system saves, depending upon the original query, one of the following,
• the incident details of all the Incident Reports raised within a particular time period
• the incident details of all the Incident Reports for the specified test identifier
• the incident details of all the incidents related to a specified Incident Report

E4.0 Management of Network Effects
The network effects, as mentioned above, are detected by analysing a number of related incidents. These network effects can be organised in a hierarchical structure. Such a structure displays the relationship between general and more specific network effects. For example, specific effects related to packet delay may include step delay variations and delay spikes. Similarly effects related to packet losses may include excessive losses and loss of service. Loss of service, in turn may be classified into short breaks and long breaks.

These effects can be organised as a tree, where the root indicates the most general effect and the leaves indicate the specific incidents related to that effect (Figure E-12). Similar structures can be used to construct a search facility over an incident report management system.
High Level Network Performance Information: Incidents and Effects

Figure E-12: Hierarchical Management of Network Effects
Appendix F

Framework for Simulating Information Systems Based on Intermediate Information

F1.0 Introduction

Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behaviour of the system or of evaluating various strategies (within the limits imposed by a criterion or a set of criteria). Thus the process of simulation involves the construction of a model and the analytical use of the model for studying the problem [Sha75].

A model is a representation of an object, system or idea in some form other than the entity itself. It aids in explaining, understanding or improving a system.

Simulations are generally considered useful when appropriate mathematical and analytical models of the problem are not available or cannot be manipulated by the investigators. These are also desirable in situations where it may be difficult to conduct experiments and observe phenomena in their actual environments. Moreover simulations are desirable when time compression is required for systems or processes with long time frames [Jai91]. Additionally simulation can aid in the verification of results obtained from the analytical models of the process under study.

The purpose of building and using a simulation may fall into one of the following categories [DavOke89],

- **comparison** of simulation runs that can be used to assess the effect of changing a decision variable. The results of different runs can be evaluated in terms of objectives.
- **prediction** of different states of the system in future, subject to assumptions regarding its present state as well as its behaviour during the course of the simulation.
- **investigation** into the behaviour of the system, rather than to perform detailed experimentation, by monitoring the simulation's response to different stimuli.
This distinction may not always be clear. Exploratory analysis may be followed by
detailed statistical analysis, comparison of different models as well as prediction of
the effects of changing different variables.

This appendix provides an introduction to the basic reusable components that can
be used to develop a simulator for an information system that employs
Intermediate Information to provide the required results. Examples of simulators
for information systems based on Intermediate Information built from these
components is also provided. A system for generating appropriate stimulus for
these simulators is explained before concluding this appendix.

F2.0 Components of an Object Oriented Simulator

An information system receives different information requests from the users. The
information system provides the users the requested information in response to
their requests. During the process the state of the information system also changes
as it attempts to cache frequently requested information that may be required by the
users (Figure F-1).

Information systems based on Intermediate Information are expected to cache
Intermediate Information processed from primitive data so that these may be
reused to provide the required information. Moreover Intermediate Information at
the finest granularity may be re-cycled to provide Intermediate Information at a
coarser level granularity (Figure F-1).

Information systems and data warehouses acquire the primitive data from the
operational systems and the data collection and acquisition systems. This data is
integrated into the information system and is pre-processed to provide the required
information (summaries, for e.g. averages, sums and counts). Thus there are two
main events occurring in the information system environment. These events are
fetch, i.e. data integration and summarisation, and query specification, i.e. user
requests for data and information.

This section describes the basic components of a simulator for an information
system that provides the required information from appropriate Intermediate
Information. This simulator has been developed to investigate the performance of
different cache replacement and prefetch algorithms that can be used to manage the
Intermediate Information cache. Prefetching is used here to describe the pre-
emptive generation of Intermediate Information elements that may fulfil future information requirements.

---

For the simulators described here it is assumed that once primitive data has been retrieved from the operational systems and integrated into the warehouse, Intermediate Information for finest granularity is computed and cached. All subsequent information elements are derived from the Intermediate Information at the required level of granularity. These Intermediate Information elements are generated from the Intermediate Information elements at the finest granularity.

Basic structure of the simulator is shown in figure F-2.

---

The time required by an information system to respond to different queries are stored in a CSV (Comma Separated Values) file. This file is represented in figure F-2 as *Response Times* file. These values are measured from a practical
implementation of the system under study. These values are stored in the following format,

\[
[\text{WindowSize}_1], [\text{MinimumTime}_1], [\text{MaximumTime}_1] \\
[\text{WindowSize}_2], [\text{MinimumTime}_2], [\text{MaximumTime}_2] \\
\vdots \\
[\text{WindowSize}_N], [\text{MinimumTime}_N], [\text{MaximumTime}_N]
\]

An object of the \texttt{ResponseTimeReader} class reads the CSV file storing the response times and generates an object of the \texttt{ResponseTime} class for each window size specified in the file. Prototype for the \texttt{ResponseTimeReader} class is as follows

```cpp
/*
 * Header file for TimeRead.cpp
 * by Omar Bashir
 */

#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#ifndef _RESPONSETIME
#include "resptime.h"
#endif
#define _RESPONSETIMEREADER 0

class ResponseTimeReader
{
    private:
        ifstream InputFile;
        int ErrorFlag;

    public:
        ResponseTimeReader(char *); //Constructor
        int getNext(ResponseTime*); //Get the next response time

};
```

The prototype for the \texttt{ResponseTime} class is shown below

```cpp
/*
 * Header file for RespTime.cpp
 * by Omar Bashir
 */

```
#include <stdlib.h>
#include <string.h>
#define _RESPONSETIME 0

class ResponseTime
{
    private:
        int WindowSize;
        int MinTime;
        int MaxTime;

    public:
        ResponseTime(char*);
        ResponseTime();
        int getWindowSize(void); //Method to return WindowSize
        int getMinTime(void);    //Method to return MinTime
        int getMaxTime(void);    //Method to return MaxTime
        void changeValues(int,int,int); //Change instance variable values
    }

Events or users' requests to the information system are provided to the system as another text file, i.e. events specification file. As mentioned before, the system responds either to query specifications or prefetch specifications. The lines in the events specification file specifying queries are coded as follows

[CurrentDate],QuerySpecs, [QueryStartDate], [QueryEndDate], [WindowSize]

The lines specifying the prefetch events in the events specification file are coded in the following format

[CurrentDate],Prefetch, [PrefetchStartDate], [PrefetchEndDate]

The event specification file can be created by recording the events occurring on a real system.

Events specification file is read by an object of the EventReader class. This object generates an object of the Prefetch class if it encounters a prefetch specification and an object of the QuerySpecs class if it encounters a query specification while reading the events specification file. The QuerySpecs class inherits from the Prefetch class.

The prototype for the EventReader class is given below

/*
   This is the header file for EvntRdr.cpp
Framework for Simulating Information Systems Based on Intermediate Information

by Omar Bashir

EvntRdr.h

*/

#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#ifndef _QUERYSPECS
#include "qryspecs.h"
#endif
#ifndef _PREFETCH
#include "prefetch.h"
#endif

class EventReader
{
private:
    ifstream InputFile;
    int ReaderHealth; // -10: File not opened
    // 0: EOF
    // 5: Incorrect format
    // 1: Prefect Event
    // 10: QuerySpecs Event
    char InputLine[100];

public:
    EventReader(char *); // Constructor
    int getEvent(); // Get the next event and return its
    // type
    QuerySpecs getQuery(); // Get the query object
    Prefetch getPrefetch(); // Get the prefetch object
};

The prototype for the Prefetch class is given below

/*
This is the header file for Prefetch.cpp

by Omar Bashir

*/

#include <stdlib.h>
#define _PREFETCH 0

class Prefetch
{
protected:
    int EventDay;
    int StartDay;
    int EndDay;

public:
    Prefetch(int, int, int); // Constructor
    Prefetch();
    Prefetch(); // Get the day of the event

    Prefetch();
int getStartDay(void); //Get the Start day
int getEndDay(void); //Get the end day of the event

The prototype for the **QuerySpecs** class is given below

```c
/*
   This is the header file for QrySpecs.cpp
   by Omar Bashir
*/
#ifndef _PREFETCH
#include "prefetch.h"
#endif
#define _QUERYSPECS 0

class QuerySpecs : public Prefetch
{
    protected:
    int WindowSize;

    public:
    QuerySpecs(int Day, int Start, int End, int Window) : Prefetch(Day, Start, End)
    {
        WindowSize = Window;
    }

    QuerySpecs():Prefetch(0, 0, 0)
    {
        WindowSize = 0;
    }

    int getWindowSize(void);
};
```

Objects of the **QueryProcessor** class simulate the operation of the Intermediate Information processing sub-system. An object of this class can be instantiated to read a specified response times file to acquire the suitable response time values. Alternately response time values can be passed as an array of objects of **ResponseTime** class to the constructor.

Objects of the **QuerySpecs** class generated by the object of **EventReader** class are passed with the pointer to an object of the **CacheFiler** class as parameters to the `processQuery()` method of the **QueryProcessor** class. The object of the **CacheFiler** class simulates a cache manager. The `processQuery()` method returns an object of the **QueryPerformance** class. This object contains information regarding the performance of the system in response to the object of the
QuerySpecs class and the status of the cache simulated by the object of CacheFiler class.

Prototype of the QueryProcessor class is given below

```cpp
/*
   This is the header file for qryprssr.cpp

   by Omar Bashir
*/

#ifndef _COMPONENTQUERYSPECS
#include "cqryspcs.h"
#endif  
#ifndef _QUERYDECOMPOSER
#include "qrybrkr.h"
#endif  
#ifndef _QUERYPERFORMANCE
#include "qryprfrm.h"
#endif  
#ifndef _CACHEFILER
#include "filer.h"
#endif  
#ifndef _RESPONSETIMERREADER
#include "timeread.h"
#endif 
#define _QUERYPROCESSOR 0

class QueryProcessor
{
    struct ResponseTimeValue
    {
        int WindowSize,MinTime,MaxTime;
    };

private:
    struct ResponseTimeValue ProcessDelay[7];
    int ProcessorHealth;
    QueryDecomposer ComponentQueryGenerator;
    void findTimes(QuerySpecs,int*,int*);

public:
    QueryProcessor(char *);
    QueryProcessor(ResponeTime DelayTimes[7]);
    QueryProcessor();
    void listResponseTimes(void);
    int getHealth(void);
    QueryPerformance processQuery(QuerySpecs,CacheFiler*);
};
```

Prototype for the CacheFiler class is given below

```cpp
/*
   This is the header file for Filer.cpp
*/

```
Framework for Simulating Information Systems Based on Intermediate Information

by Omar Bashir

Filer.h

#include <fstream.h>
#ifndef _COMPONENTQUERYSPECS
#include "cqryspcs.h"
#endif
#include <string.h>
#define _CACHEFILER 0

class CacheFiler
{
    struct CachedFileName
    {
        int Day;
        int Hour;
        int Window;
    };

    private:
    fstream CacheFile;
    int CacheHealth;

    public:
    CacheFiler();
    CacheFiler(char*);
    ~CacheFiler();
    int getFile(ComponentQuerySpecs); //Find if the file is
    //available in the
    //cache or not. Return -1
    //if not
    int putFile(ComponentQuerySpecs); //Put a file in the cache,
    //return -1 on failure
    ComponentQuerySpecs getNext(int*); //Get next file from the
    //cache. -1 on eof
    int dumpCache(char*); //Dump the contents of the cache file in
    //a text file, -1 on error.
    void gotoBOF(void); //Go to the top of the file
    int getHealth(void); //Get CacheHealth
};

Objects of the QueryProcessor class use an object of the QueryDecomposer class to decompose the objects of the QuerySpecs class (composite queries) into objects of ComponentQuerySpecs class (component queries), one for each analysis window. A trigger to the getNext() method of an object of the QueryDecomposer class provides the subsequent object of the ComponentQuerySpecs class. If this method is triggered after the last object of the ComponentQuerySpecs class for a specific object of the QuerySpecs class has been returned to the calling method, this method returns -1.
The object of ComponentQuerySpecs class can be passed to the `getFile()` method of the object of the CacheFiler class. If the cache manager detects that an indication of a file containing Intermediate Information relevant to the object of this ComponentQuerySpecs class is not present in the cache, then the `getFile()` method returns -1 to the calling method. An object of the ComponentQuerySpecs class can be passed to the `putFile()` method of the object of the CacheFiler class. This method simulates the creation of the file of Intermediate Information related to the object of the ComponentQuerySpecs class in the cache.

The prototype for the ComponentQuerySpecs class is given below

```cpp
/*
 * This is the header file for the component query specification
 * by Omar Bashir
 *
 */

#ifndef _QUERYSPECS
#include "qryspecs.h"
#endif
#define _COMPONENTQUERYSPECS

class ComponentQuerySpecs:QuerySpecs
{
    private:
    int CurrentHour;

    public:
    ComponentQuerySpecs(int Day, int Hour, int Window)
        : QuerySpecs(Day, 0, 0, Window)
    {
        CurrentHour = Hour;
    }
    ComponentQuerySpecs():QuerySpecs()
    {
    }
    int getDay(void);
    int getHour(void);
    int getWindowSize(void);
};
```

The prototype for the QueryDecomposer class is given below

```cpp
/*
 * This the header file for query decomposer object
 * by Omar Bashir
 *
 */

#ifndef _COMPONENTQUERYSPECS
#include "qrybrkr.h"
#endif
```
Thus, every object of the \texttt{QuerySpecs} class passed to the \texttt{processQuery()} method of an object of the \texttt{QueryProcessor} class is decomposed to relevant objects of the \texttt{ComponentQuerySpecs} class by an object of the \texttt{QueryDecomposer} class. Each object of the \texttt{ComponentQuerySpecs} class represents query specification for a specific analysis window during the period specified in the associated object of the \texttt{QuerySpecs} class. An object of the \texttt{CacheFiler} class determines whether Intermediate Information for each object of the \texttt{ComponentQuerySpecs} class is present in the cache or not. If Intermediate Information for a specific object of the \texttt{ComponentQuerySpecs} class is not available in the cache, it is included in the cache.

If the desired Intermediate Information is not present in the cache, it is assumed that the system will take maximum response time for the specified analysis window size to process that component query. The response time to process each component query is passed as parameter to \texttt{addTime()} method of the object of \texttt{QueryPerformance} class, along with the information regarding the availability of the associated Intermediate Information in the cache. Once all the component queries of a composite query have been processed, the object of the \texttt{QueryPerformance} class is returned to the calling method. This object can be saved in a results file (CSV format) with the help of an object of \texttt{ResultsSaver} class.

Prototype for the \texttt{ResultsSaver} class is given below.
This is the header file for rsltsvr.cpp
by Omar Bashir
*/

#include <fstream.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#ifndef _QUERYPERFORMANCE
#include "qryprfrm.h"
#endif
#define _RESULTSSAVER 0

class ResultsSaver
{
private:
    ofstream ResultsFile;
    int ResultHealth;

public:
    ResultsSaver();
    ResultsSaver(char *);
    ~ResultsSaver();
    int getHealth(void);
    int saveResults(QueryPerformance);
};

The prototype for the QueryPerformance class is given below.

/*
This is the header file for qryprfrm.cpp
by Omar Bashir
*/
#ifndef _QUERYSPEC
#include "qryspecs.h"
#endif
#define _QUERYPERFORMANCE 0

class QueryPerformance
{
private:
    int TotalTime;
    int MaxTime;
    int TotalAccesses;
    int Hits;
    int Misses;
    QuerySpecs QueryValue;
    void initValues(void);

public:
    QueryPerformance(QuerySpecs);
    QueryPerformance();
    QuerySpecs getQuerySpecs(void);
};
F3.0 Examples of Simulator Systems
This section illustrates the application of the objects described in section F2.0 in
developing simulators for information systems that derive the required information
from suitable Intermediate Information elements. The first example discusses a
simulator for a basic information system that neither employs any prefetching nor
any cache replacement mechanisms. In such information systems, any Intermediate
Information elements generated as a response to a query to the information system
are cached for subsequent reuse. The system indefinitely retains these Intermediate
Information elements.

The second example illustrates the convenience of developing an information
system simulator that employs prefetching mechanism. This is accomplished by
simply adding a new object in the basic information system simulator. This object
performs appropriate prefetching operations for the information system. For this
example, the prefetch object employs a simple informed prefetching mechanism,
where the prefetch object is informed regarding the Intermediate Information
elements to be pre-processed.

F3.1 Basic Information System Simulator
Intermediate Information at the finest granularity is used to construct
Intermediate Information at coarser levels of granularity expeditiously. It is
therefore not considered to be a part of the cache. As the primitive data is
retrieved from the data collection systems and integrated into the information
system, Intermediate Information at the finest granularity is also computed
from these data and stored accordingly.

The architecture for an information system simulator shown in figure F-2 can
be expanded into the basic information system simulator. This architecture is
shown in figure F-3.

The information system simulator, upon initialisation, instantiates an object of
the ResponseTimeReader class. This object reads the CSV file containing the
response time specifications for the information system and generates objects
of the \texttt{ResponseTime} class. These objects are used to initialise an object of the \texttt{QueryProcessor} class.

![Diagram of Basic Information System Simulator](image)

Figure F-3 : Basic Information System Simulator

The system also instantiates objects of the \texttt{CacheFiler}, the \texttt{EventReader} and the \texttt{ResultsSaver} classes. The object of the \texttt{EventReader} class is used to read the events file and passes every object of the \texttt{QuerySpecs} class to the object of the \texttt{QueryProcessor} class. The pointer to an object of the \texttt{CacheFiler} class is also passed to the object of \texttt{QueryProcessor} class. The object of \texttt{QueryProcessor} class generates an object of the \texttt{QueryPerformance} class which specifies the performance of the information system in response to the query specification. This object of the \texttt{QueryPerformance} class is stored in the results file by the object of the \texttt{ResultsSaver} class.

F3.2 Information System Employing Informed Prefetching

This simulator attempts to simulate the operation of an information system that employs informed prefetching to pre-process the required Intermediate Information elements. This would allow investigation of the characteristics of such information systems.

Simulator for an information system employing informed prefetching can be implemented by introducing an object of the \texttt{InformedPrefetcher} class in the simulator for basic information system. This object attempts to simulate the generation of the Intermediate Information elements of specified analysis window sizes as the primitive data is collected from the data collection entities and is integrated into the information system. This allows for expeditious
processing of queries that request information based on the pre-processed Intermediate Information elements. The prototype for the InformedPrefetcher class is shown below.

```cpp
/*
 * This is the header file for the infpfc.h
 * by Omar Bashir
 */

define _INFORMEDPREFETCHER 0

class InformedPrefetcher
{
    private:
        ifstream InFile;
        int Numbers2Fetch;
        int Windows2Fetch[7];

    public:
        InformedPrefetcher();
        InformedPrefetcher(char*);
        int prefetchData(Prefetch, CacheFiler*); // -1 if unsuccessful
          // 0 if successful
          // -10 if not prefetched as not
          // specified
        void listWindows2Fetch(void);
};
```

Objects of the InformedPrefetcher class, upon instantiation, read a file containing the analysis window sizes for which Intermediate Information elements need to be pre-processed. Parameters for the prefetch operation, i.e. start and end time are passed as an object of the Prefetch class and a pointer to an object of CacheFiler class as parameters to the prefetchData() method. For each analysis window size specified, the object of the Prefetch class is converted into an object of the QuerySpecs class. This object of the
QuerySpecs class is decomposed into appropriate objects of the ComponentQuerySpecs class by an object of the QueryDecomposer class. The object of the CacheFiler class uses these objects of the ComponentQuerySpecs class to update the cache status file, thus indicating the presence of the specified Intermediate Information elements.

The architecture of the simulator for the information system employing informed prefetching to pre-process the required Intermediate Information elements is shown in figure F-4.

F4.0 Simulating Requests to the Information System

As mentioned previously, the stimulus to the information system simulator are coded as events in text files. An object of the EventReader class in the information system simulator reads these event specification files. This object in the information system simulator then produces the relevant event objects, i.e. of the Prefetch or the QuerySpecs class, indicating the occurrence of the corresponding events.

The stimulus file for the information system simulator can be generated by a program that can adequately simulate the generation of requests by the users of the system. It has been suggested that information requests made to data warehouses and information systems are mostly regular queries that retrieve information required to generate reports. Some users generate queries that may appear to be random in nature and are directed at investigating specific events and patterns of interest in the historical data [InmWWW], [Gla97].

Figure F-4: Information System Employing Informed Prefetching
This section describes a trace simulator that attempts to simulate the generation of query and prefetch specifications in an information systems environment. The architecture of the trace simulator program is shown in figure F-5.

![Trace Simulator Diagram](image)

**Figure F-5 : Trace Simulator**

The trace simulator program requires the user to specify
- the name of the regular events specification file
- the name of the file to write trace output
- two long integer values as seeds to generate random queries
- the end time in days of simulation

The regular events specification file provides the parameters for regular events. This file contains the specification for the partitions of the trace that are generated by a regular pattern of system requests. Each partition of the trace represents the specification of the trace during a particular period of simulated time and a partition differs from another only on the basis of the specification of parameters of events. Format of this file is shown in the following example

```plaintext
[TRACE TIME]
0,10.
[PREFETCH]
7.
[QUERY]
7,3.
28,24.
[QUERY END]
[TRACE END]
[TRACE TIME]
11,30.
```
The line after [TRACE TIME] represents the start and end times of the specific portion of the trace. The line after [PREFETCH] represents the gap (in days) between successive prefetch events. Lines between [QUERY] and [QUERY END] contain specifications for query events. These include the gap (in days) between the associated query specification event and the analysis window size. One partition of a trace can contain 10 such query specifications. Specifications for a particular portion of the trace are terminated by [TRACE END] specifier. The end of the file contains the file termination string, i.e. ***.

The trace simulator program instantiates an object of the SystemTimer class at the time value of 0. This object advances the simulation time as days once its tick() method is triggered. This method also provides the current simulation time value as day. The prototype for the SystemTimer class is given below.

```
// TIMER CLASS
class SystemTimer
{
    private:
        int CurrentTime;

    public:
        SystemTimer();
        int tick(void);
};
```

The program also instantiates an object of the ofstream class (OutFile) which is used by relevant objects to store events in the trace file. If an error occurs while attempting to open the output trace event file, the system exits after printing an appropriate error message.
This program uses an object of the `RegularEventGenerator` class which opens the prescribed regular event specification file upon instantiation. If an error occurs while attempting to open the regular event specification file, the program exits after displaying an appropriate error message. If the object detects an unexpected end of file or the termination string, this object returns the control to the main program which exits after displaying an appropriate error message. The prototype for the `RegularEventGenerator` class is given below.

```cpp
/*
   This is the header file for revntgen.cpp

   by Omar Bashir

*/

#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#define _REGULAREVENTGENERATOR 0

class RegularEventGenerator
{
  struct QueryData
  {
    int DateSpecs,WindowSize;
  };

  private:
  ifstream InFile;
  int GeneratorHealth; // -1 if file error on opening
                         // -10 if file error on reading
                         // 0 Hunky Dory
  int TraceStart,TraceEnd;
  int PrefetchDateSpecs;
  int NumberOfQueries;
  struct QueryData Queries[10];
  // Private Methods
  int readSpecs(void); //Returns -1 when file terminator read
  void readPrefetch(void);
  void readQueries(void);
  void readTraceTimes(void);
  void constructQuery(int,char*);
  void showQueries(void);

  public:
  RegularEventGenerator();
  RegularEventGenerator(char*);
  ~RegularEventGenerator();
  int getHealth(void);
  int getEvent(int,ofstream*); //Returns -1 when file terminator read
};
```
As the program loops, the `tick()` method of the object of `SystemTimer` class is triggered to retrieve the current date. The date is passed to the object of the `RegularEventGenerator` class along with the `OutFile` object. The object of the `RegularEventGenerator` class checks if the date provided by the clock is within the date specifications of the trace partition.

If the date value is within the date specifications of the trace partition, the object of the `RegularEventGenerator` class checks if the date value corresponds to generation of a suitable event. This object then generates the suitable event specifications and uses the `OutFile` object to store these specifications to the trace file on the disk. Prefetch events have precedence over the query specification events. If the date value is not within the date specification of the trace partition, the object of the `RegularEventGenerator` class attempts to read the next partition of the trace.

If the date is greater than 1, it is passed to an object of `RandomQueryGenerator` class along with the `OutFile` object. This object of the `RandomQueryGenerator` class is instantiated by passing two seeds to the constructor to initialise as many random number generators. These random number generators are implemented as objects of the `RandomNumberGenerator` class. These objects generate two single digit pseudo-random numbers. If one random number is greater than the other, the object of the `RandomQueryGenerator` class generates a random query specification for a random period before the current date value. The difference of the two single digit random numbers is used as the analysis window size for the query specification event. Architecture of the `RandomQueryGenerator` class is shown in figure F-6. The prototypes of the `RandomQueryGenerator` class is given below.

```c++
/*
This is the header file for rqrygen.cpp
by Omar Bashir
*/

#include <fstream.h>
#include <string.h>
#include <stdlib.h>
#ifndef RANDOMNUMBERGENERATOR
#include "randgen.h"
#endif
#define RANDOMQUERYGENERATOR 0

class RandomQueryGenerator
```

257
Framework for Simulating Information Systems Based on Intermediate Information

private:
RandomNumberGenerator NormalOne;
RandomNumberGenerator Threshold;
void generateQuery(char*, int, unsigned long);
int getWindowSize(unsigned long);
int getDate(int);
void makeQueryString(int, int, int, int, char*);

public:
RandomQueryGenerator();
RandomQueryGenerator(unsigned long, unsigned long);
void getQuery(int, ofstream *);

Figure F-6: Architecture of the RandomQueryGenerator Class

F5.0 Summary and Discussion

This appendix explains some of the objects that can be used to expeditiously construct simulators for information systems employing Intermediate Information to process the required information elements. Other objects can be included into the basic simulator to analyse the performance of these information systems when suitable Intermediate Information prefetching and cache replacement mechanisms are used.

Stimuli to these information simulators are provided via text files containing the prefetch and query specifications. These text files can be generated by monitoring the activities of the users on an implemented information system. Alternately these text files can be automatically generated by simulating the behaviour of the users of these systems. User of information and decision support systems can be broadly categorised into two classes. Users belonging to one class interact with the information system regularly in a predictable manner. Users belonging to the other category post queries to the information systems to investigate different phenomena of interest. These interactions generally appear to be random in nature.
Results generated from a simulation session from each of the simulators discussed in this appendix are explained here. The stimulus for these simulation sessions was generated by simulating user interaction with the system. The regular events were generated on the basis of the following specification.

[TRACE TIME] 0.15.
[PREFETCH] 7.
[QUERY] 7, 3.
28.24.
[QUERY END]
[TRACE END]
[QUERY] 7, 4.
14, 12.
[QUERY END]
[PREFETCH] 7.
[TRACE END]
[TRACE TIME] 29, 50.
[PREFETCH] 7.
[QUERY] 7, 6.
[QUERY END]
[TRACE END]
[TRACE TIME] 50, 64.
[PREFETCH] 7.
[QUERY] 7, 4.
21, 6.
[QUERY END]
[TRACE END]
***

The events specifications generated by the events simulator on the basis of the above mentioned specifications are shown in table F-1. Random query specifications are also included in this stimulus file.

The response times for the information system being studied are listed in table F-2.
Framework for Simulating Information Systems Based on Intermediate Information

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Type</th>
<th>Start Date</th>
<th>End Date</th>
<th>Window Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Prefetch</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>QuerySpecs</td>
<td>0</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>QuerySpecs</td>
<td>2</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>14</td>
<td>Prefetch</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>QuerySpecs</td>
<td>7</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>QuerySpecs</td>
<td>2</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>QuerySpecs</td>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>QuerySpecs</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>21</td>
<td>Prefetch</td>
<td>14</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>QuerySpecs</td>
<td>14</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>24</td>
<td>QuerySpecs</td>
<td>6</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>28</td>
<td>Prefetch</td>
<td>21</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>QuerySpecs</td>
<td>21</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>QuerySpecs</td>
<td>14</td>
<td>28</td>
<td>12</td>
</tr>
<tr>
<td>31</td>
<td>QuerySpecs</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>33</td>
<td>QuerySpecs</td>
<td>8</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>35</td>
<td>Prefetch</td>
<td>28</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>QuerySpecs</td>
<td>28</td>
<td>35</td>
<td>6</td>
</tr>
<tr>
<td>35</td>
<td>QuerySpecs</td>
<td>23</td>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>36</td>
<td>QuerySpecs</td>
<td>6</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>QuerySpecs</td>
<td>3</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>42</td>
<td>Prefetch</td>
<td>35</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>QuerySpecs</td>
<td>35</td>
<td>42</td>
<td>6</td>
</tr>
<tr>
<td>42</td>
<td>QuerySpecs</td>
<td>21</td>
<td>42</td>
<td>24</td>
</tr>
<tr>
<td>44</td>
<td>QuerySpecs</td>
<td>5</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>46</td>
<td>QuerySpecs</td>
<td>17</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>49</td>
<td>Prefetch</td>
<td>42</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>QuerySpecs</td>
<td>42</td>
<td>49</td>
<td>6</td>
</tr>
<tr>
<td>50</td>
<td>QuerySpecs</td>
<td>6</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>QuerySpecs</td>
<td>23</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>55</td>
<td>QuerySpecs</td>
<td>12</td>
<td>43</td>
<td>4</td>
</tr>
<tr>
<td>56</td>
<td>Prefetch</td>
<td>49</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>QuerySpecs</td>
<td>49</td>
<td>56</td>
<td>4</td>
</tr>
<tr>
<td>57</td>
<td>QuerySpecs</td>
<td>23</td>
<td>48</td>
<td>4</td>
</tr>
<tr>
<td>63</td>
<td>Prefetch</td>
<td>56</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>QuerySpecs</td>
<td>56</td>
<td>63</td>
<td>4</td>
</tr>
<tr>
<td>63</td>
<td>QuerySpecs</td>
<td>42</td>
<td>63</td>
<td>6</td>
</tr>
<tr>
<td>63</td>
<td>QuerySpecs</td>
<td>28</td>
<td>35</td>
<td>4</td>
</tr>
<tr>
<td>64</td>
<td>QuerySpecs</td>
<td>6</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>65</td>
<td>QuerySpecs</td>
<td>1</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>67</td>
<td>QuerySpecs</td>
<td>5</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>68</td>
<td>QuerySpecs</td>
<td>6</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>69</td>
<td>QuerySpecs</td>
<td>23</td>
<td>48</td>
<td>24</td>
</tr>
<tr>
<td>70</td>
<td>QuerySpecs</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Table F-1: Event Specification for Simulation

The simulator of the information system using informed prefetching prefetches Intermediate Information with analysis window sizes of 3, 4 and 6 hours.
The response times for each of the query specifications generated by both the simulations are shown in figure F-7.

<table>
<thead>
<tr>
<th>Analysis Window</th>
<th>Minimum Response Time</th>
<th>Maximum Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>42</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>24</td>
<td>45</td>
<td>135</td>
</tr>
</tbody>
</table>

Table F-2: Response Times for the Simulation

Similarly the cache miss rates for each query specifications obtained from both these simulations can be seen in figure F-8.

It can be seen from these graphs that, for this particular simulation session, the cache miss rates as well as the response times obtained by employing informed prefetching mechanism are lower as compared to those expected from an information system that does not employ any Intermediate Information prefetch mechanism. However to achieve some substantial results, these simulations need to be conducted repeatedly with varying parameters and the results generated by these simulations need to be statistically analysed.
Figure F-8: Simulated Cache Miss Rate