Performance Controls for Distributed Telecommunication Services

Conor J McArdle B Eng

This thesis is submitted in partial fulfillment of
the requirements of the degree of

PhD in Electronic Engineering

November 2004

Thesis Supervisor
Professor Thomas Curran

Dublin City University
School of Electronic Engineering

Acknowledgments

I wish to thank my supervisor Professor Thomas Curran for his help and guidance during the course of the PhD Thanks also to Rob Brennan and Brendan Jennings for their help and encouragement I would also particularly like to thank my wife Annemarie and all my family and friends for their love and support over my years of study

Thesis Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of *PhD in Electronic Engineering*, is entirely my own work and has not been taken from the work of others save to the extent that such work has been cited and acknowledged within the text of my work

Signed Conor McArdle ID No 97971146

Date 21st January 2005

Table Of Contents

Table of Figures

Table of Tables

Performance Controls for Distributed Telecommunication Services

Conor J McArdle

Abstract

As the Internet and Telecommunications domains merge, open telecommunication service architectures such as TINA, PARLAY and PINT are becoming prevalent Distributed Computing is a common engineering component in these technologies and promises to bring improvements to the scalability, reliability and flexibility of telecommunications service delivery systems This distributed approach to service delivery introduces new performance concerns As service logic is decomposed into software components and distributed across network resources, significant additional resource loading is incurred due to inter-node communications This fact makes the choice of distribution of components in the network and the distribution of load between these components critical design and operational issues which must be resolved to guarantee a high level of service for the customer and a profitable network for the service operator

Previous research in the computer science domain has addressed optimal placement of components from the perspectives of minimising run time, minimising communications costs or balancing of load between network resources This thesis proposes a more extensive optimisation model, which we argue, is more useful for addressing concerns pertinent to the telecommunications domain The model focuses on providing optimal throughput and profitability of network resources and on overload protection whilst allowing flexibility in terms of the cost of installation of component copies and differentiation in the treatment of service types, in terms of fairness to the customer and profitability to the operator Both static (design-time) component distribution and dynamic (run-time) load distribution algorithms are developed using Linear and Mixed Integer Programming techniques An efficient, but sub-optimal, run-time solution, employing Market-based control, is also proposed

The performance of these algorithms is investigated using a simulation model of a distributed service platform, which is based on TINA service components interacting with the Intelligent Network through gateways Simulation results are verified using Layered Queuing Network analytic modelling Results show significant performance gains over simpler methods of performance control and demonstrate how trade-offs in network profitability, fairness and network cost are possible

List of Abbreviations and Acronyms

AE	Application Entity
ANSI	American National Standards Institute
API	Application Program Interface
ASN 1	Abstract Syntax Notation One
BILP	Binary Integer Linear Program
CCF	Call Control Function
CO	Computational Object
CORBA	Common Object Request Broker Architecture
CSM	Communication Session Manager
CTI	Computer Telephony Integration
DPE	Distributed Processing Environment
ETSI	European Telecommunications Standards Institute
FIFO	First In First Out
GA	Genetic Algorithm
GW	Gateway
GWSN	Gateway Service Node
IA	Initial Agent
IDL	Interface Definition Language
IN	Intelligent Network
INAP	Intelligent Network Application Part
INCM	Intelligent Network Conceptual Model
IOP	Interoperability Protocol
IP	Intelligent Peripheral
ITU	International Telecommunications Union
JIDM	Joint Inter-domain Management
LAN	Local Area Network
LP	Linear Program
LQN	Layered Queuing Network
LQNS	Layered Queuing Network Solver
MIP	Mixed Integer Program
MOL	Method of Layers
MSC	Message Sequence Chart
MVA	Mean Value Analysis
NCCE	Native Computing and Communications Environment
NMF	Network Management Forum
NP	Non-determmistic Polynomial
ODL	Object Definition Language
OMG	Object Management Group

ORB	Object Request Broker
OSL	Optimisation Solutions Library
PA	Provider Agent
PINT	PSTN Internet Interworking
PSTN	Public Switched Telephone Network
PT	Percentage Thinning
QoS	Quality of Service
ROS	Remote Operation Service
RPC	Remote Procedure Call
RR	Rejection Ratio
RTT	Round Trip Time
SC	Service Component
SCEF	Service Creation Environment Function
SCF	Service Control Function
SCFP	Service Control Function Proxy
SCP	Service Control Point
SDF	Service Data Function
SDP	Service Data Point
SF	Service Factory
SLA	Service Level Agreement
SMF	Service Management Function
SN	Service Node
SRF	Service Resource Function
SRVN	Stochastic Rendezvous Network
SS 7	Signalling System No 7
SSF	Service Switching Function
SSM	Service Session Manager
SSP	Service Switching Point
TCAP	Transaction Capabilities Application Part
TCSM	Terminal Communication Session Manager
TINA	Telecommunications Information Network Architecture
TINA-C	Telecommunications Information Network Architecture Consortium
UA	User Agent
UAP	User Application
UQP	Unconstrained Quadratic Binary Program
USM	User Service Session Manager
VPN	Virtual Private Network

Chapter 1. Introduction

Chapter 1 introduces the research area of this thesis The thesis goals are stated in §1 2 and an overview of the remainder of the thesis is given in §1 3

1.1. Overview of the Research Area

As the Internet and Telecommunications domains merge, open telecommunication service architectures are becoming prevalent Traditional service delivery systems, such as the Intelligent Network, are moving towards more open service architectures such as TINA, PARLAY and PINT Distributed Object Computing is a common engineering component for these technologies and promises to bring improvements to the scalability, reliability and flexibility of service delivery systems However, a distributed approach to system design introduces new performance concerns As service logic is decomposed into software components and distributed across network resources, significant additional resource loading is incurred due to inter-node communications This makes the choice of distribution of components in the network and the distribution of load between these components critical design and operational issues which must be resolved to guarantee a high level of service for the customer and a profitable network for the service operator

Telecommunications service systems and software normally operate to very stringent performance criteria as system downtimes and overloads are expensive occurrences for the service operator Rejected service attempts during such events cause a direct loss of revenue to the operator In addition, service agreements with business customers often specify that any revenue loss to the service subscriber, due to system non-performance, is to be reimbursed by the operator Apart from direct revenue losses, service subscribers expect a high level of service from telecommunications networks As competition in the domain increases, the quality of service delivered to the customer has become a more important issue Customers now differentiate between service providers based on quality of service and so loss of customers to alternative service providers, due to poor service performance, is a growing business concern

Control of system performance and protection of system revenues pose particular technical challenges for the operator Telecommunication service networks are normally protected with robust performance control mechanisms In particular, much effort has been focused on load control for traditional Intelligent Network service platforms (*Service Control Points*) When tightly integrated service platforms are replaced with more open distributed systems implementations, it becomes a more challenging problem to provide the same high level of performance control Protection of nodes from overload and assurance of high system throughput and revenue in such environments, are more difficult goals to achieve than in traditional centralised systems due to the increased complexity of multiple service components interacting over multiple network nodes

Much research has already been done in the area of general distributed systems performance, however, performance objectives for general-purpose distributed systems differ from those of dedicated telecommunications systems Distributed systems performance optimisation normally focuses on minimising delays or inter-node communications time in the system This is commonly a design-time problem with the objective being that, under average traffic patterns at run-time, request processing times may be as short as possible However, absolute speed is generally not the main concern for processing of telecommunications services Human user interaction times determine maximum allowable delays and these times are typically long compared to computer processing speeds Once delay is bounded to an acceptable (human) level, of more concern are protection from overloads, minimisation of the rejection rate of user requests and maximisation of system throughput Recently, revenue optimisation has also become a direct objective of performance controls As a wider range of diverse service offerings is introduced to a service platform, it is often deemed important to differentiate between service types, giving more important services prioritised access to system resources

In general-purpose distributed systems, coupled with design-time performance considerations, simpler dynamic load sharing schemes are implemented to account for changes in traffic patterns at run-time Currently, dynamic controls generally assume simplified interactions between software components Although software design tools are increasing in capabilities, and can now produce detailed execution timing information for an application, this information is normally only used for deciding application partitioning and static assignment of components to network nodes and is not used for design of run-time load controls With the need for tighter performance control in telecommunications systems it may be of advantage to leverage detailed application execution information to produce more effective run-time performance controls

In order that distributed service platforms deliver the promised advantages of ease of software design, software reuse and flexibility and potentially greater scalability and higher performance, it is necessary to consider the performance of the underlying distributed system from a telecommunications perspective This will involve applying telecommunications performance sensibilities to the performance methodologies that have emerged from the general computer science domain In this thesis, we investigate both the traditional telecommunications and the more general computer science approaches towards performance control Drawing on ideas from both domains, we propose performance approaches for emerging and future telecommunications service networks

1.2. Thesis Aims and Objectives

The overall goal of the work undertaken in this thesis is to develop a set of controls and techniques for assuring performance of telecommunication services executing in a distributed object computing environment Ideas are to be tested and evaluated though service platform simulations, verified with analytic results The detailed aims and objectives for the thesis are listed below

- Investigation of literature in the area of load control techniques for traditional telecommunication service platforms
- Investigation of existing performance improvement techniques for general-purpose distributed systems
- Identification of prevalent performance concerns for distributed platforms in a telecommunications environment and identification of desirable performance control features for such systems
- Development and analysis of suitable design-time optimisation for assignment of software components to network nodes
- Development and analysis of suitable run-time load distribution and load control approaches
- Consideration of the following in relation to proposed control schemes

  System throughput and revenue maximisation

  System overload protection

  Multiple service type systems

  Consideration of distributed system communications costs

  Generality of approach and applicability to future networks

- Investigation of scenarios for distributed computing platforms for telecommunications services, in particular in relation to Intelligent Network evolution

- Construction of a detailed model of a representative distributed service platform to allow analysis of proposed approaches and comparison to existing methods
- Verification of the simulated system implementation with an analytic model

1.3. Thesis Organisation

Chapter 2 (*Background and Literature Review*) gives an introduction to the evolution of telecommunications service platforms towards distributed systems Discussed are IN, TINA and IN/CORBA inter-working The area of performance control in IN and TINA is reviewed Next, an introduction to distributed systems, from a performance perspective, is given We review literature in the area of optimising performance of distributed systems, namely scheduling, task allocation, load sharing, load balancing, admission control, overload control and component allocation Having considered the literature, requirements for performance controls are identified and possible approaches to our problem area are considered The chapter concludes with specific objectives for performance controls considered in the thesis

Chapter 3 (*Methods and Tools*) reviews mathematical and software tools encountered in the thesis, namely, probability and queuing theory, analysis of product-form and non product-form networks of queues, simulation methods for network performance analysis, Linear Programming, Mixed Integer Programming, and Market-based problem formulations

Chapter 4 (*Model of a Distributed Telecommunications Service Platform*) develops a model of a distributed telecommunications service platform and describes the simulation model and test services used for assessment of performance control techniques developed in the thesis

Chapter 5 (*Computational Object Allocation and Performance Control Strategies*) develops a general set of performance control strategies for telecommunications services operating in distributed object environments The approaches proposed are based on the system performance requirements identified in Chapter 2 A method for optimal allocation of software components to network nodes, which considers component copy installation costs and network revenue maximisation, is presented Optimal run-time controls are developed A sub-optimal market-based algorithm is also developed as a run-time control

Chapter 6 (*Analysis of Service Platform and Performance Controls*) investigates the efficiency of our proposed methods developed in Chapter 5 and draws comparison with existing methods An analytic model is developed, using *Layered Queuing Networks*, and the accuracy of the simulator is verified

Chapter 7 (*Conclusions and Future Work*) assesses our work in terms of our original aims and objectives and more generally in terms of contributions to this area of research Potential improvements on the work are identified and possible further research is suggested

Chapter 2. Background and Literature Review

This chapter gives an overview of the evolution and future of telecommunications services. Related technologies and initiatives, namely Intelligent Networks, TINA, CORBA, IN/CORBA and TINA/IN inter-working, are reviewed. The state-of-the-art in performance control for Intelligent Networks and TINA is examined and related literature in the area of general distributed systems performance is reviewed. The necessity for effective performance control of future distributed service platforms is identified and the requirements for such performance control mechanisms are discussed. The related work to-date in this area is reviewed and the need for further work is identified. Finally, we define detailed research objectives for the thesis.

2.1. Introduction

Over recent years, telecommunication service and system design has seen much change. The introduction of open and distributed architectures has become prevalent in both research and industry. Distributed architectures have been seen as attractive as they promote a separation of concerns, allowing services and the network to be treated independently.

Telecommunication systems are becoming more complex due to continuing increases in power and functionality. Many technological areas such as computing technology, information technology, network management, integration with the Internet and Web servers are involved in service development and deployment. As new technologies and elements are introduced into telecommunication services, to resolve particular problems or to introduce new service features, an increase in complexity and incompatibility has resulted. As more and more services are introduced, deployment and inter-working of new and existing services has become a resource intensive and time-consuming problem. Effort has been focused on these issues, detracting from the actual effort devoted to developing new services.

Driven by these factors, there is an increasing orientation in the telecommunications society towards an open software creation and standard computing environment Service Architectures based on middleware technologies, such as CORBA, are increasingly seen as the appropriate infrastructure in a value-added telecommunications network The reasons for this move towards the adoption of middleware technology include

- the increased ability to cope with system scalability issues,
- the ability to leverage commercial off-the-shelf IT technologies,
- the advantages of an open standards process of middleware such as CORBA,
- the ease of system integration with existing working systems,
- the ability to leverage new technologies as they emerge and
- the avoidance of technology and vendor lock-in

Although the object-oriented, distributed computing model provided by middleware promises to be beneficial for expanding currently deployed service systems such as the Intelligent Network, by increasing scalability, reliability, flexibility and providing interfacing to other service networks such as the Internet, fundamental questions have been identified as requiring careful investigation

- software and data partitioning,
- performance control overload control and load distribution and
- middleware performance

In the past, much effort has been concentrated on providing effective load control and overload protection schemes for nodes in existing Intelligent Network systems Similar efforts are required if new middleware based systems are to succeed in providing similar or greater levels of system reliability and efficiency and if the benefits offered by distributed object computing technologies are to be maximised In the following sections of this chapter we give an overview of current and future telecommunication service architectures and then review the approaches that have been taken towards performance

2.2. Telecommunication Service Architectures

Early telecommunication services were embedded into the call switching network which typically consisted of a hierarchy of switches, e g a local exchange level, an intermediate exchange level and a transit exchange level When services were situated at the transit (top) level, there was a large overhead incurred for their use as a large number of switches and related trunks needed to be accessed in order to use a service For this reason, services were

migrated to lower levels of the hierarchy, reducing overhead In the extreme case, each local exchange level switch contained the service logic and data meaning that every service must be loaded into every switch's software before it could be used Thus service maintenance and addition was very difficult, especially as the number of services contained in each switch increased Consequently, the addition of new services occurred very rarely As a single company was responsible for running an exchange and all of the services it offered there was no competitive market for service provision since the company running the exchange was the only service provider Lack of competition led to lack of innovation, and so service provision did not progress [Harju, 1994]

2 2 1. The Intelligent Network

To resolve these issues, the Intelligent Network (IN) was developed in the 1980s The IN concept was to separate the service processing from the switches so as to ease and speed the deployment of new services and reduce the then escalating complexity of exchanges There was also a desire to share service data, distribute processing among dedicated service network elements so as to meet an increasing demand for more sophisticated telecommunication services and allow for scalability The intention of IN was also to standardise interfaces and protocols so as to enable an open platform for uniform service creation, implementation and management allowing multiple service vendors to participate in a competitive market IN standardisation has taken place in ANSI [1999], ETSI [1994] and the ITU-T [1997] and has been widely deployed

A general framework for creating international standards for INs, known as the *Intelligent Network Conceptual Model* (INCM), was developed to provide a framework for the design and description of the target IN architecture [ITU-T, 1993a] As a self-contained model, it captures the whole engineering process of the IN At the functional level of the INCM model, services are implemented by *functional entities* (Figure 2 1) These functional entities are realised as corresponding *physical entities* (normally corresponding to network nodes) in the *physical plane* The categories of functions and their corresponding physical entities can be differentiated as

Basic call-handling functions The *Connection Control Function* (CCF) resides at the switch and provides the functionality for basic call processing In the physical plane, the *Service Switching Point* (SSP) provides the platform for running the CCF

Service execution functions The *Service Switching Function* (SSF) contains the logic for controlling switch resources during execution of a service It also provides a service-independent interface to the *Service Control Function* (SCF) which controls network resources during service execution In the physical plane the SSP provides the platform for

running the SSF and the *Service Control Point* (SCP) provides the platform for the SCF The *Service Data Function* (SDF) contains both customer-related and network-related data and provides standardised access methods, enabling the SCF to use this data The SDF is implemented on the *Service Data Point* (SDP) in the physical plane The SRF provides service-related functions such as collecting dialled digits or playing service announcements The SRF is normally implemented by an *Intelligent Peripheral* (IP) in the physical plane In addition to the call-related functional entities, the SMF and SCEF provide management and service creation functions

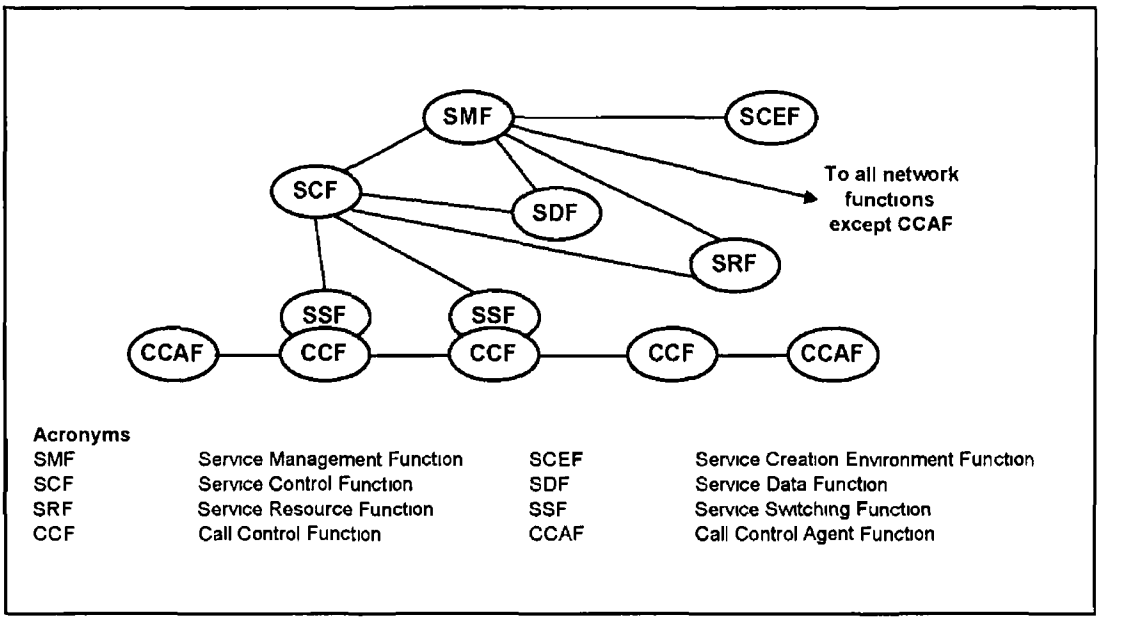


Figure 2 1 Functional Entities in the IN Distributed Functional Plane

By way of example, an invocation of an IN service starts by the detection of a *trigger event*, at a predefined point within the call For example, dialling a 1800 number will trigger a service for translating the dialled digits into a 'real' telephone number When a trigger is detected, normal call processing is suspended, and a query is sent to the SCP The query is expressed as an *Intelligent Network Application Part* (INAP) message sent to the SCP over the signalling network, the *Signalling System No 7* (SS 7) [ITU-T, 1993] The SCP processes this query and can either return a set of instructions to the SSP at the switch, or execute its own service logic, possibly communicating with other entities such as the SDP and IP In the case of the 1800 number example, the SCP queries the SDP for the 'real' number and returns it to the SSP for call routing in the switch

Although IN is a huge step forward from early embedded service logic schemes and has enjoyed a large degree of success, it is still limited by a number of deficiencies [Blair *et al*, 2001] IN services have limited user interfaces Service processing is dependent on the

detection of a trigger in the context of a call prior to connection set-up i e services are invoked for the end user by the transport provider Within the switch, the call is modelled by a Basic Call State Model, which is strongly telephony oriented and inflexible The SCP has a standardised, generic, connection view of the call processing resources an IN switch offers This standard model, while enabling some switch vendor independence, offers little in the way of transport technology independence

The physical separation of the SSP and the SCP, attempted to provide the reliability required for telecommunication services by provisioning expensive centralised facilities IN did not attempt to distribute the service itself by endeavouring to build a reliable system by redundantly deploying relatively cheap and unreliable facilities, or by any other means This was reasonable given the state of the art in distributed systems at the outset of IN standardisation, but ultimately it leaves the IN looking like a legacy centralised system, by its nature more prone to catastrophic failure than a counterpart with distributed intelligence

While the importance of management aspects was acknowledged, the IN initiative concentrated standardisation effort on service switching and control It was only later that telecommunication service architectures recognised the need to support the concurrent development of services and their management facilities

However, while failing to achieve a complete logical separation between a service and its underlying communications technology, IN nevertheless established a physical separation between a service and its delivery that provided a useful basis for further service modelling work

2 2 2 TINA

In 1993 the Telecommunications Information Network Architecture Consortium (TINA-C) was founded The consortium aimed to define a Telecommunications Information Networking Architecture (TINA) which would enable the efficient introduction, delivery and management of telecommunication services beyond that provided by the IN [TINA-C, 1997] Due to the rapid convergence of telecommunications and computing, the focus of attention moved away from the physical network to software-based systems Essentially TINA provides a set of concepts and principles for specification, design, implementation, operation and management of software systems for telecommunication networks, with a view to leveraging the advantages of object oriented design methodologies and distributed object platform technologies The use of object-oriented principles for service modelling is expected to improve the interoperability of services allow the re-use of software and to allow flexible deployment strategies for software in the network The use of distributed middleware platforms, such as the *Common Object Request Broker Architecture* (CORBA) [Schmidt,

1997], allow hiding of distribution concerns from applications The types of services supported by TINA range from voice-based services to multi-media, multiparty services

In the TINA service architecture, a service is described as a set of interacting objects called *Service Components* (SCs) Each SC consists of one or more *Computational Objects* (COs), which are executed in a *Distributed Processing Environment* (DPE) The DPE shields the services from the distributed nature of the system, taking care of communication between objects and maintaining location and communication transparency in the system The DPE in turn rests upon the *Native Computing and Communications Environment* (NCCE), the native systems software that controls a hardware platform A schematic view of TINA from a computational viewpoint is shown in Figure 2 2 below

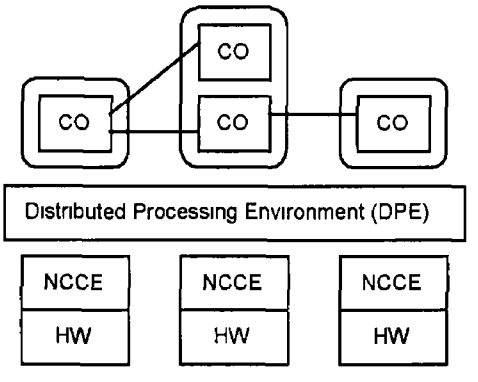


Figure 2 2 A Schematic View of the Different Layers of TINA

The *Computational Object* (CO) in TINA is the main entity at the functional level encapsulating service data and functionality Each service consists of a number of COs interacting through their prescribed interfaces Some COs are service specific, while others are common TINA components COs reside in different *domains*, according to which role they take in the service (Figure 2 3) *User Domain* objects are either users of services or otherwise closely tied to the user whilst *Provider Domain* objects are related to service provisioning

TINA Computational Objects are organised in terms of the *session* concepts to which they relate in TINA, namely, the *Access Session*, *Service Session* or *Communication Session*

2 2 2 1 Access Session Related Computational Objects

User Application (UAP) The UAP is the user domain representation of a service application A UAP allows a user to both create and join existing sessions UAPs belong to the User Domain

Provider Agent (PA) The PA is a service independent CO and is defined as the user's end-point of an access session It allows the setup of trusted relationships between a user and a

provider, by interaction with an *Initial Agent* (described below) During an access session a PA conveys requests to and from the user to the rest of the system PAs belong to the User Domain

Initial Agent (IA) The IA is both a user and service independent CO that is defined as the initial access point to a domain It has the capability to set up trusted relationships between domains by interaction with a PA These access sessions can be either anonymous or not, in the former case the *User Agent* (below) is accessed as an anonymous user agent IAs belong to the Provider Domain

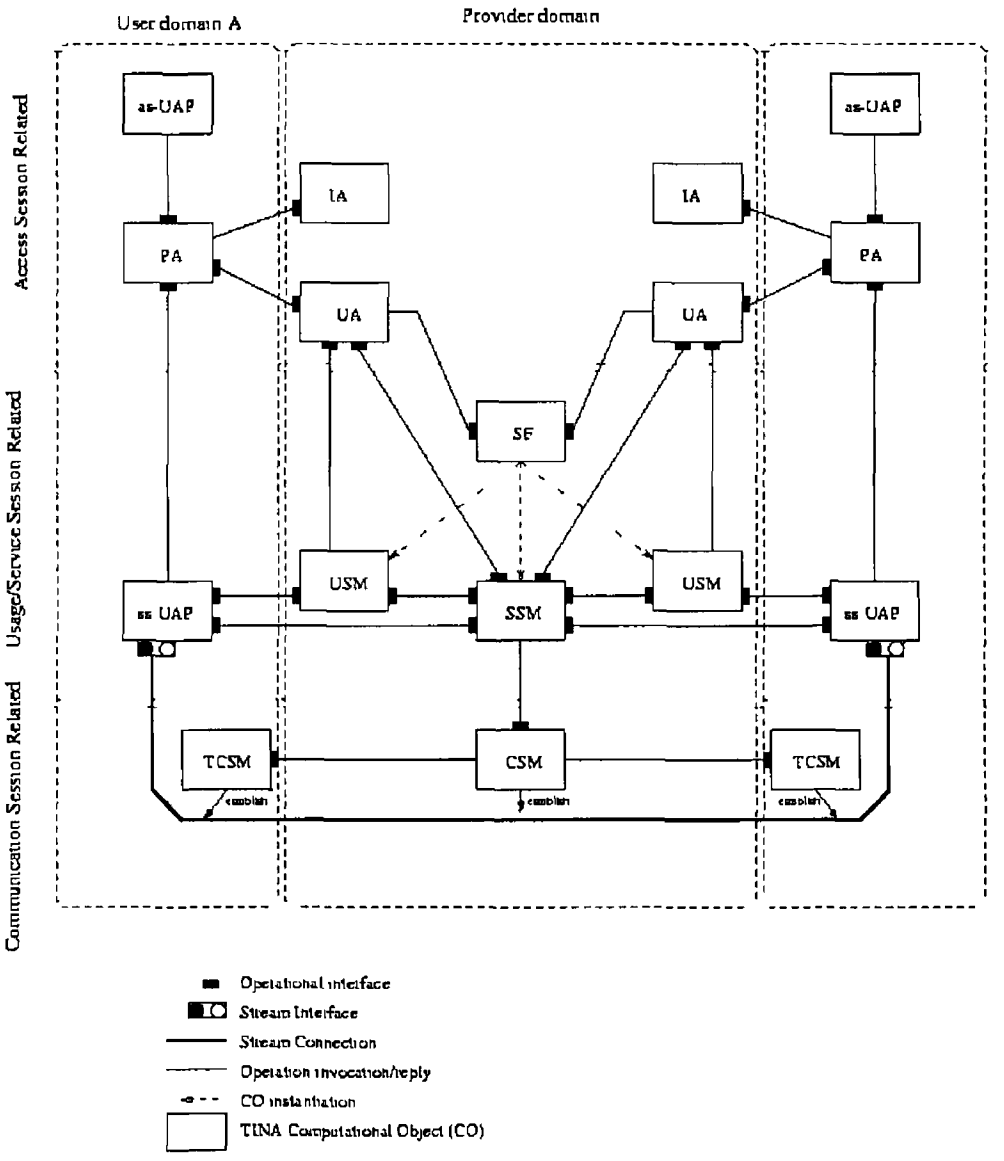


Figure 2 3 Relationships between COs and Domains

User Agent (UA) The UA represents the user in the provider's domain It is the provider domain's end-point of an access session with a user and is accessible from the user's domain regardless of that domain's location A UA acts on the behalf of the user, and may be seen as

an intelligent agent-like component. UAs reside in the provider domain. UAs can be both *Named* and *Anonymous* (both subtypes inherit the properties of the UA). Named UAs are used when a user is a subscriber to the provider's domain, and can be used for further authentication if needed. Anonymous UAs are used when a user does not or can not disclose its identity to a provider. An anonymous UA might for instance be used when calling from a phone booth or other places where user identity cannot be completely assured.

2.2.2.2. Service Session Related Computational Objects

Service Factory (SF): A service specific CO that can create service session components for a specific service type. It also assembles the resources necessary for the existence of a component that it creates. The SF resides in the Provider Domain.

User Service Session Manager (USM): The USM is a service specific CO that contains information about service capabilities, that are local to a user. For instance it keeps track of local resources used by a user. In the case of suspension and resumption of a service session, the USM maintains the local state for a user. It resides in the Provider Domain.

Service Session Manager (SSM): The SSM is a service specific CO that contains the service-specific and generic session control logic for a service. An SSM supports services that are shared among users in a session. In the case of suspension of a service, the SSM maintains the state of the session until it is activated again. The SSM supports accounting and resides in the Provider Domain.

2.2.2.3. Communication Session Related Computational Objects

Communication Session Manager (CSM): The CSM is a service independent CO that manages end-to-end stream bindings between stream interfaces. It resides in the Provider Domain.

Terminal Communication Session Manager (TCSM): The TCSM is a service independent CO that manages the local intra-node connections in the user's domain. It answers to requests from a CSM to setup, modify or remove stream connections. It resides in the Provider Domain.

2.2.3. TINA in Use

Numerous papers and reports have been published on TINA since the architecture was introduced in 1995. Sebastiano *et al.* [1998] provide a survey on how TINA service architectures and distributed processing platforms may be used to develop third-generation mobile systems. Alexandre *et al.* [1999] discuss how mobility could be incorporated into TINA services. Juan *et al.* [1998] present a brokerage architecture, which focuses on the

development of electronic commerce in TINA environments Several authors have also advocated the TINA component model for replacing the Intelligent Network SCP, for example in [Herzog & Magedanz, 1997] and [Mampaey, 2000] Capellmann [2000] reports on prototypes for TINA based SCPs inter-operating with the Intelligent Network through an *Adaptation Unit* An OMG standard [OMG, 1999] has already been defined for inter-working of IN and CORBA which would facilitate building of such Adaptation Units in a standard way Mampaey [2000] discusses the benefits of such an approach for IN and states that a TINA-based IN can offer standardised and service-independent interfaces to prevalent technologies, such as Computer Telephony Integration (CTI) applications and Internet-based applications such as Internet call waiting He sees a generic TINA framework as providing inter-working in a structured way across different technologies (Figure 2 4)

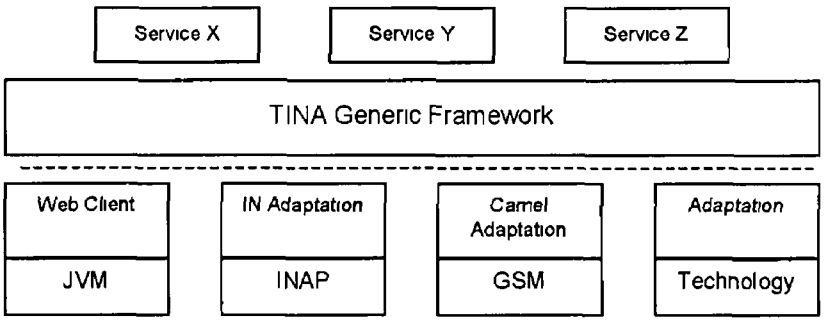


Figure 2 4 TINA-based Interworking

Although the uptake of complete end-to-end TINA solutions has been slow, it can be seen in the literature that the elements of the TINA service architecture have merits for structuring of general service provisioning platforms In Chapter 4, we propose a model consisting of the components of the standardised IN/CORBA gateway (acting as the Adaptation Unit) interacting with a set of TINA Service Components, which provide the functionality of the Intelligent Network SCP

We will next examine the underlying motivation for use of Distributed Object Computing for telecommunication service provisioning platforms and give details of the IN/CORBA inter-working methodology Firstly, we give a brief overview of CORBA before examining IN-CORBA inter-working in detail

2.3. Overview of CORBA

The *Common Object Request Broker Architecture* (CORBA) is an open *Distributed Object Computing* infrastructure, standardised by the *Object Management Group* (OMG) [OMG, 1995] CORBA automates many common network programming tasks such as object registration, location, and activation It also manages error-handling and parameter

marshalling and demarshalling Figure 2 5 illustrates the main components of the OMG *Reference Model Architecture*

Object Services - These are domain-independent interfaces that are useful to any type of CORBA application For example, a service providing for the discovery of other available services is almost always necessary regardless of the application domain Two examples of Object Services that fulfil this role are (1) The Naming Service, which allows clients to find objects based on names, (2) The Trading Service, which allows clients to find objects based on their properties

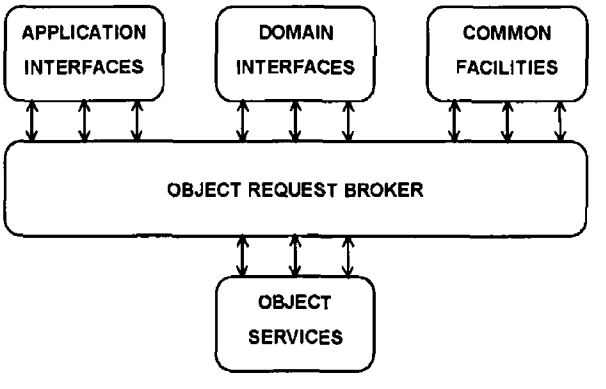


Figure 2 5 OMG Reference Model Architecture

Common Facilities – Common Facilities are oriented towards end-user applications An example of such a facility is the *Distributed Document Component Facility* (DDCF), a compound document Common Facility based on OpenDoc

Domain Interfaces - These interfaces fill roles similar to Object Services and Common Facilities but are oriented towards specific application domains For example, one of the first OMG Domain Interfaces was the *Product Data Management* (PDM) enablers for the manufacturing domain Other OMG Domain Interfaces have been defined in the telecommunications, medical, and financial domains

Application Interfaces - These are interfaces developed specifically for a given application Because they are application-specific, and because the OMG does not develop applications (only specifications), these interfaces are not standardised The main elements of the CORBA architecture are shown in Figure 2 6

Object Implementation - This defines operations that implement a CORBA IDL interface Object implementations can be written in a variety of languages including C, C++, Java, Smalltalk, and Ada

Client - This is the program entity that invokes an operation on an object implementation Accessing the services of a remote object should be transparent to the caller Ideally, it should

be as simple as calling a method on a local object The remaining components in the architecture help to support this level of transparency

Object Request Broker (ORB) - The ORB provides a mechanism for transparently communicating client requests to target object implementations The ORB simplifies distributed programming by de-coupling the client from the details of the method invocations This makes client requests appear to be local procedure calls When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller

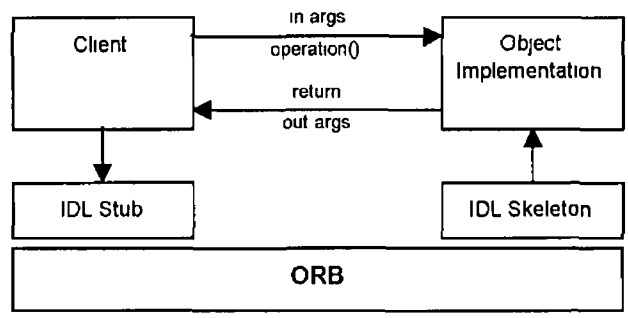


Figure 2 6 Object Request Broker Architecture

CORBA IDL Stubs and Skeletons - CORBA IDL (*Interface Definition Language*) stubs and skeletons serve as the "glue" between the client and server applications, respectively, and the ORB The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimisations

2.4. Interworking Networks and IN/CORBA

As mentioned in the previous section, two software technologies are candidates for extensive use in the telecom market (i) object-oriented design and programming, which well suits the need for reusable, open components for telecom applications, and (ii) distributed processing, which suits application to the highly distributed nature of telecom systems This has led to the definition of a middleware software layer enabling telecom-tailored distributed processing based on an object-oriented approach

Capellmann [2000] notes that the adoption of IT technologies in Intelligent Network systems, has already occured Current SCP platforms from many vendors are already structured following a client/server model, in which different UNIX based servers are connected through high-speed data networks, with front end and back end distributed computing Distributed processing environment standards, such as CORBA, are already implemented for service

management and are under evaluation for real-time applications within IN and wireless network elements

Much of the formal investigation into the future application of CORBA to IN systems has been initiated by the Eurescom P508 project [Eurescom, 1997], the goal of which was to determine the options for evolving from legacy systems towards TINA in a graduated manner Possibilities for migration from current control and management architectures to TINA had been previously investigated One major result was that the gradual introduction of TINA DPE technology, i e CORBA technology enhanced with real-time capabilities, into the existing environments, represents the fundamental prerequisite for such an evolution Particularly, the evolution of Intelligent Networks was an important study item of the P508 project In the course of these studies, two White Papers [OMG, 1996] and [OMG, 1997] have been produced by the Object Management Group's *Telecom Domain Task Force*, in order to support the emerging OMG work activities on IN/CORBA interworking These White Papers are targeted at providers of information technology solutions and have the purpose of stimulating their interest towards telecommunication operator specific needs They analyse a small subset of the problem area the introduction of middleware into the Intelligent Network

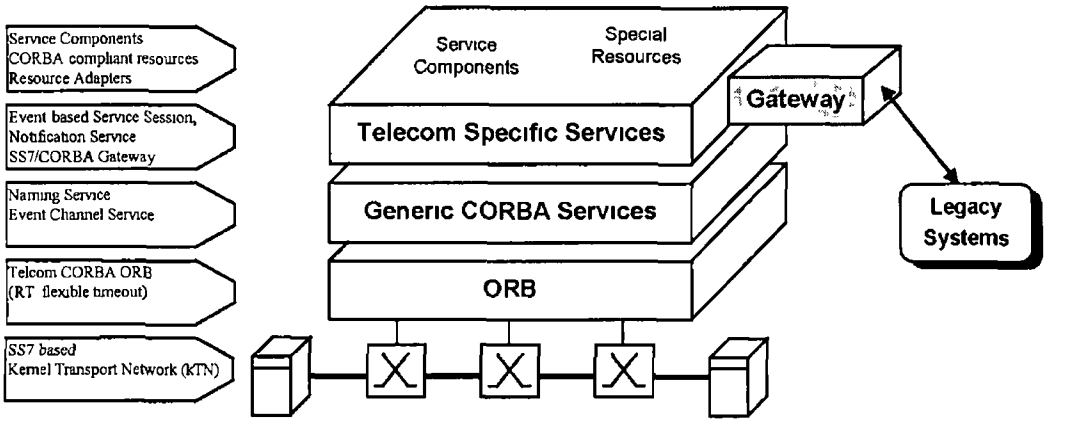


Figure 2 7 IN and CORBA - The P508 Vision

The central idea put forward in [OMG, 1997] is that of introduction of a middleware software layer into application and data servers, and eventually also into switching systems, enabling a component-based, distributed intelligence replacing the traditional monolithic Intelligent Network functional entities The switched transport network ensures transfer of user information across connections, on which calls are established The middleware platform enables realisation of IN functions in a distributed way, that is by a set of application and data servers interacting via the platform corresponding to IN functional entities This platform is based on CORBA whereby, CORBA servers containing CORBA objects act as reusable

service components Communications at the middleware platform level is based on an Interoperability Protocol (IOP) [OMG, 1995] and the application-level signalling network, ensuring communication among platform nodes, is termed the Kernel Transport Network (kTN) It should rely on the existing SS 7 signalling network [ITU-T, 1993], which fulfils important requirements for telecom applications, such as high reliability Interoperability with legacy IN elements and services is ensured by a CORBA to SS 7 gateway

On top of the middleware layer, three types of entities are deployed (1) Service / Service feature components - the service logic, implementing a wide range of services and service features by means of reusable components, (2) CORBA-compliant special resources - resources, such as bridges, databases and so on that have been designed in such a way that they can be directly plugged in on the middleware layer, (3) Adapters for special resources that interact with the exterior with a different paradigm than that of the distributed computing middleware, for example, with proprietary protocols

Taking IN legacy systems into account leads to the issue of defining which profile of the protocol stack must be used, and of building a gateway between the IN and the CORBA domains based on SS 7

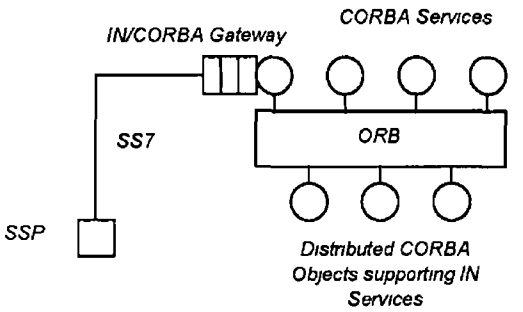


Figure 2 8 Gateway to CORBA based Service Platform

In order to enable CORBA objects to control IN SSPs it is necessary to define a dedicated application-level gateway object that provides a CORBA/IDL interface (API) to the objects on the CORBA side and an SS 7 interface towards the signalling network on the other side (Figure 2 8) This means that an IN SSP communicates with other network entities using SS 7/INAP, while on the CORBA side invocations of IDL interfaces are used for communication The gateway, located in between, is in charge of transparently adapting both types of communication The most likely realisation of such a gateway suggested by the P508 report [Eurescom, 1997] is a TCAP/CORBA gateway This is a generic application-level gateway defined for all TCAP or ROS (Remote Operation Service) Users (of which INAP is one example) by providing translation algorithms for converting between ROS constructs defined in Abstract Syntax Notation One (ASN 1) and the corresponding CORBA constructs using IDL

Once the gateway from IN to CORBA is defined, it is possible to introduce CORBA as the distributed processing platform into the intelligent layer to enable the service logic to be structured as service components, which correspond to objects on the CORBA platform This component-based approach enables dynamic service composition, i e the flexible assembling of pre-existing components to form a particular service

2 4 1 OMG IN/CORBA Interworking Specification

Interest in these OMG white papers led to the completion of an IN/CORBA inter-working specification The primary design goal of the specification [OMG, 1999] is to provide interworking mappings and supporting CORBA services that enable traditional IN systems, whose interfaces are defined using the ASN 1-based Intelligent Network Application Part (INAP) and use the SS 7 protocol stack for communication These are to inter-work with CORBA-based implementations of IN systems, whose interfaces are defined in OMG IDL and use the OMG-defined protocols for communication The interworking mappings produce IDL for a CORBA object model in the CORBA domain that provides interfaces to legacy IN systems from the CORBA domain and also provides interfaces to CORBA-based IN applications to legacy IN systems This object model may be used to build a gateway, which provides protocol conversion and alignment of execution semantics between the IN and CORBA domains, allowing full IN-CORBA interworking Supporting CORBA objects are defined by the specification that allow application naming, addressing, location and instantiation in the two domains to be aligned

2 4 2 Components of the IN/CORBA Gateway and CORBA-based SCP

The application interworking described above may be categorised into Specification Translation and Interaction Translation The specification translation is an extension of the JIDM specification translation specification [X/OPEN, 1995] which has been adopted by The Open Group / NMF JIDM defines mappings for ASN 1 basic constructs to OMG IDL The extensions allow full translation of further ASN 1 constructs, used to define INAP, into OMG IDL Interaction Translation is provided by a set of CORBA interfaces, which support the run-time interactions between CORBA-based IN implementations and legacy IN implementations Figure 2 9 shows the major interfaces defined and how they interact to provide an interworking function (gateway) between the IN and CORBA domains

In Figure 2 9, a "legacy" SSP interacts with a CORBA-based service implementation using the IN/CORBA object model Note that only interactions initiated from the IN domain are shown here although the model proposed is general and may also support interactions that are initiated from the CORBA domain The objects shown in grey are CORBA objects whose interfaces are defined in OMG IDL in accordance with the Interaction and Specification

Translations specified by the standard The Gateway Administration object (GWAdmin) performs the functions of name translation and object location between the two domains Messages arriving from a legacy SSP are addressed to a particular SS 7 Application Entity (AE), identified by a particular AE title The GWAdmin provides an interface for translating the AE title to the CORBA object reference of a Service Factory object, which may create instances of the Service Interface Object

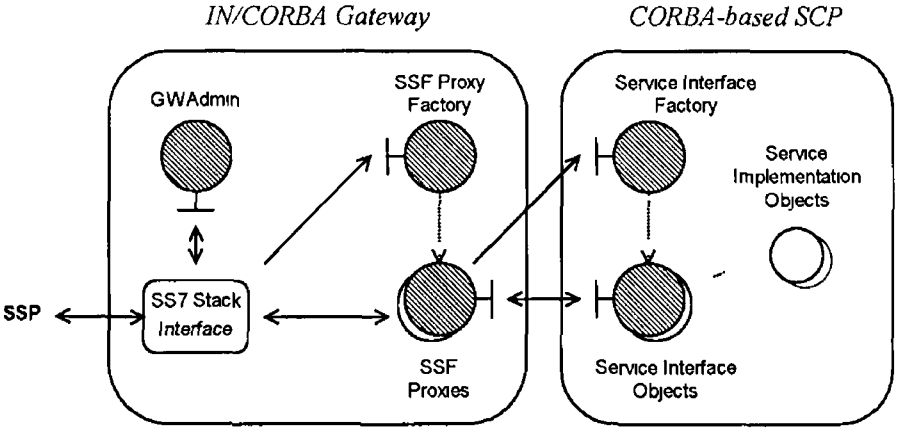


Figure 2 9 The Interworking Gateway

In order to represent the SSP in the CORBA domain, a SSF Proxy object is required This object provides an IDL interface for invocation of INAP operations on the SSF from the CORBA domain and performs the protocol translation and communication with the SS 7 stack The SSF Proxy Factory provides a standardised means of instantiating a SSF Proxy The Service Interface Object provides a complementary IDL interface for invocation of INAP operations from the SSF to the CORBA-based SCP Protocol translation for these invocations is provided in the gateway The association between SSF Proxies and Service Interfaces is maintained implicitly by a particular implementation of the gateway - one instance of a Service Interface Object may be used for several instances of a service session or each new service instance may create a new Service Interface Object The same is true for SSF Proxies This design provides implementation flexibility in terms of scalability and distribution

2.4 3 IN/CORBA Gateway and TINA Service Components

In Chapter 4, we will propose a CORBA-based SCP model derived from the IN-CORBA Gateway Components and the TINA Service Components This model will form the basis for our investigations in this thesis Several authors have also advocated the TINA component model for replacing the SCP in the IN (for example Herzog & Magedanz [1997] and Mampaey [2000]) through use of an IN-TINA *Adaptation Unit* Although many proprietary Adaptation Units have been proposed, we have chosen to base our component model on the standard IN/CORBA gateway

Having considered past, present and future architectures for telecommunication service provisioning, we next turn our attention to the related performance issues

2.5. Performance Control of Telecommunication Services Networks

All telecommunication systems have a finite capacity that limits the volume of requests that they can successfully process at any one time If the usage of the network by end-users exceeds its capacity, an *overload* condition can occur During long periods of overload, service requests join long queues at busy network processors As a result, service response times can become unacceptably long This problem may be compounded by end-users abandoning service attempts after a long wait and issuing new service requests As messages from earlier abandoned attempts remain in processor queues and new reattempts are made, the offered load to processors grows even more, leading to even higher abandonment rates and resulting in an unstable positive feedback scenario The number of service attempts that actually complete service (and earn money for the operator) decreases rapidly and the network becomes less profitable

Thus overloads are highly undesirable for both the end-user and the network operator Therefore, it is essential that steps are taken to minimise the impact overload has on network performance *Performance Control* refers to strategies and mechanisms used to manage network traffic so that network resources are efficiently used and service completion rates are maximised in all traffic conditions, and in particular during overload A broad spectrum of load controls have been proposed and implemented for telecommunication service networks, particularly in relation to the Intelligent Network where the focus is mainly on protecting the Service Control Point (SCP) from overload

Developers of future service networks consider Distributed Processing Environments as a suitable paradigm for provisioning of new services In this case, a user's service request is no longer executed on a single SCP node but is distributed across multiple processing nodes where processing at each node is required to complete servicing of the request In this scenario, an overload occurring on any one processor in the system can cause a *bottleneck* in service execution Long delays at this node alone may cause the user to abandon the service request However, to further compound the problem, processing that has already been completed for this service request on other nodes is then 'wasted', as a reattempt by the service user will require the same processing to be repeated In this respect, performance control of individual nodes in a distributed environment is even more critical than in an Intelligent Network, as one poorly performing node can impact heavily on a system that may be otherwise only lightly loaded Many solutions to balancing of load to prevent such

bottlenecks have been proposed in the area of general distributed systems research However, very few solutions have been investigated specifically for telecommunication services and we propose that this is a worthwhile area of research

When we consider inter-working of the Intelligent Network and a distributed service platform, we propose that we must also consider performance control at network interfaces as well as load balancing internally in the distributed system An IN/CORBA gateway in this scenario behaves very much like a SCP, when viewed from the Intelligent Network side Thus, it is natural that the gateway should accurately represent the load situation in the entire distributed system as if it were a SCP This allows existing Intelligent Network performance control mechanisms to operate normally without the need for knowledge of the load situation on individual nodes in the distributed system Thus, we contend that Intelligent Network performance controls could provide overload protection for the distributed platform as a whole Performance controls operating through such gateways have not been studied, as far as we are aware, and we propose that research in this area is a worthwhile endeavour We contend that it is desirable that the *internal* performance controls (load balancing in the distributed system) and *external* performance controls (at the gateway) should be integrated and co-ordinated in an optimal way to provide the most benefit to the service user and network operator That is, the performance controls should, at the very least, keep system response times at an acceptable level and maintain a profitable system for the operator We propose other desirable properties of the performance control later in this chapter (in §2 7)

In the remainder of this section (§2 5) we consider existing work in the area of Intelligent Network performance control, as it impacts on functioning of performance controls at the gateway and generally reveals the thinking and methodologies behind performance control from a telecommunications perspective We contend that methods in Intelligent Network performance control may also be of use in distributed systems We also review the literature in the area of performance control in TINA as it impacts on performance controls in our scenarios In the next section (§2 6) we review the broader area of distributed systems performance in detail, as this impacts greatly on optimising performance controls for distributed service platforms

2.5 1 Load Control in Intelligent Networks

In a typical Intelligent Network scenario, multiple Service Switching Points (SSPs) communicate with a single Service Control Point (SCP) during service execution (Figure 2 10) As the SCP is the central controller and executor of service logic, its protection against overload has been studied widely IN specifications mandate only a minimal degree of load control functionality and equipment vendors and network operators are therefore afforded a good deal of freedom when implementing IN load controls

Numerous approaches have been proposed and compared in the literature The approaches taken can be broadly categorised into two types, *active* and *reactive* [Lodge, 2000] In active strategies the SSPs detect SCP overload based on local measurements, such as response delay of messages sent to the SCP If an overload condition is detected, new traffic to the SCP is *throttled* (reduced or limited by some means) until SCP overload abates In reactive strategies the SCP detects overload itself, by means of an *overload detection* algorithm and notifies the SSPs of its overload status The SSP then implements a load throttling algorithm until it is notified by the SCP that overload has abated

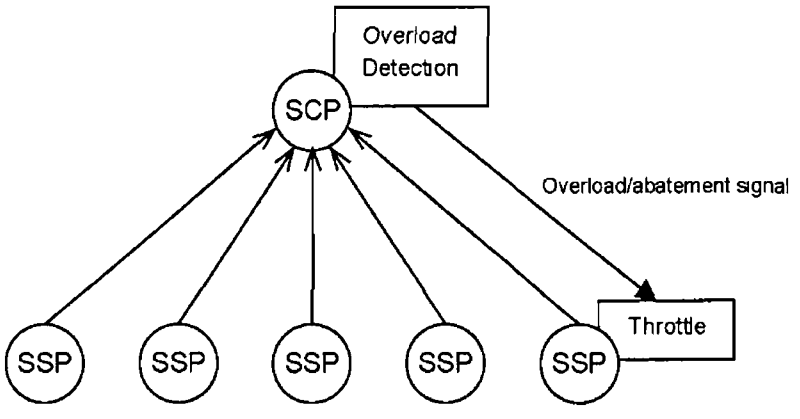


Figure 2 10 Overload Detection and Throttling in an Intelligent Network

Most research, on single SCP scenarios, has focused on reactive strategies and differs mainly in terms of the types of overload detection and load throttling algorithms used A range of SCP overload detection algorithms, making use of different performance metrics, have been proposed and investigated

- **Queue Length Control** when the SCP input queue length exceeds a threshold value the SSPs are notified of an onset of an overload condition
- **Processor Utilisation** The proportion of time the SCP central processor spends on processing service-related messages is measured or estimated over a set interval and compared to threshold values When employed, the aim is typically to keep utilisation in a pre-defined range, or close to but below some target capacity
- **Incoming Message/Session Rate** The number of messages or new sessions arriving at the SCP over a set interval is compared to a threshold
- **Average Response Delay** The average time spent by messages in the SCP, from placement in the input queue to the end of processing, is measured over a set interval and compared to a threshold
- **Dropped Messages** SCPs typically drop messages if they have been in the input queue for a longer than specified time The number of dropped messages over an interval is compared to a threshold value to indicate an overload condition

Note that these metrics are used for notifying the SSPs of both detection of onset of an overload and abatement from an overload condition

Jennings [2001] makes some observations on the relative pros and cons of controls based on these metrics Queue length control has the advantage of reacting very quickly to overload onset but can lead to unnecessary over-control caused by random fluctuations in arrival rates Session rate control assumes that session counts will be directly proportional to the session's processing requirements, however this will not be the case if the SCP supports heterogeneous services, each with different processing requirements He also notes that response delay schemes suffer from the same difficulties in heterogeneous service He sees dropped messages and processor utilisation metrics as giving better defined performance objectives

A number of load throttling algorithms that reduce the acceptance rate of IN service requests in response to SCP overload indications have also been investigated, of which the most common are

Percentage Thinning With percentage thinning a specified proportion of requests arriving during a time interval are accepted The decision as to whether or not a particular request is accepted can be based on Bernoulli trials (refer to §3 1 4), where the probability of success is the Percentage Thinning coefficient indicated by the SCP

Call Gapping This limits the number of requests accepted in a certain interval to a specified number The throttle operates by enforcing a minimum time spacing between call acceptances where no additional requests can be accepted while the gap timer is active Various call gapping based schemes have been proposed for example in [Pham and Betts, 1992], [Smith, 1995], [Hac and Gao, 1998]

The relative merits of load throttles have been examined by Kihl and Nyberg [1997], Lodge [2000] and Jennings [2001] The general consensus is that Percentage Thinning is more dynamic than Call Gapping, in that the Percentage Thinning coefficients are dynamically computed to provide the necessary reduction in expected traffic during the coming control interval In general it is found that both throttles are approximately equal in terms of protecting the SCP, albeit under the assumption that Call Gapping gap intervals are appropriate to the particular network structure Percentage Thinning is seen to exhibit fair treatment of users, because all SSPs throttle the same proportion of traffic, regardless of their size For the same reason Percentage Thinning is a scalable throttle not only is it independent of SSP size, it is also independent of the number of SSPs in the network On the other hand, Lodge [2000] notes that a significant advantage of Call Gapping is that it places a strict upper limit on the number of accepted sessions and therefore is not susceptible to sudden increases in arrival rates, as is Percentage Thinning

2 5.2. Network-Centric IN Load Control

Although the above schemes can be deployed in multiple SCP scenarios, they function so that each SCP protects itself independently without regard for the load situation in other parts of the network There has been recent interest in network-centric, as opposed to node-centric, approaches to load control, which aim to optimise loading across the whole network

Lodge [2000] has formulated a network-centric strategy that involves the formulation of Linear Programming problems (refer to §3 3 2), whose solution defines the optimal threshold values to be used by a Percentage Thinning load throttle residing at SSPs The optimisation involves the maximisation of generated service revenues subject to load constraints on SCPs and SSPs, as well as constraints to ensure that pre-defined weightings (similar to priorities) between service types are reflected in the Percentage Thinning coefficients These weightings are representative of the relative importance of successfully setting up a session of a given service type in comparison to a session of other service types In the strategy specification, weights are calculated using information regarding service session revenue, processing requirements and service level agreements Specifically the weight of service type j at resource x (SSP or SCP), denoted ω_{xj}, is given by

$$\omega_{xj} = \frac{R_j q_j e_{xj} \mu_{xj}}{\sum_{j=1}^{J} R_j q_j e_{xj} \mu_{xj}}$$

where R_j is the set-up revenue associated with service type j, q_j is the numerical quality-of-service level of service type j, e_{xj} is the number of messages in a type j service that are processed by resource x, μ_{xj} is the service rate of service type j messages at resource x and J is the number of services supported by resource x Quality-of-service levels are arbitrarily set by the network operator, on the basis of factors such as acceptable delays, customer importance or financial penalties associated with non-adherence to service level agreements

The use of agent technology for multi-SCP networks has been investigated by Patel *et al* [2000] They describe a multi-agent system realising an artificial computational market in which the processing capacity of SCPs is 'sold' to SSPs in a manner that maximises global utility, which in this case is generated profit Davidsson *et al* [2000] describe an approach based on mobile broker agents, which sell SCP processing capacity to SSPs on an autonomous basis, that is, not in the context of an auction (§3 4 gives a summary of market-based control techniques)

Jennings *et al* [1999] derived a multi-service globally optimal co-operative market strategy for controlling load in a multi-SCP network Similar to Lodge [2000], each service has a

particular revenue generating capacity and a profit optimal solution is sought. Arvidsson *et al.* [1997] have also considered profit optimal congestion control in INs based on an estimate of round trip delay and using baysian decision theory for solution.

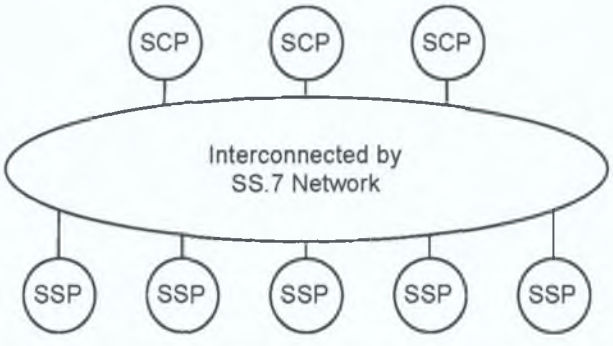


Figure 2.11: Multi-SCP Intelligent Network

We describe [Jennings *et al.*, 1999] here as some of the ideas are used as a basis for formulating our market-based strategy in Chapter 5. In this strategy, load control in a multi-SCP IN is carried out by means of tokens, which are 'sold' by 'providers' (the SCPs) and 'bought' by 'consumers' (SSPs). The amount of tokens sold by a SCP controls the load offered to it and the amount of tokens bought by a SSP determines how many service requests it can accept. 'Trading' of tokens in an 'auction' is carried out such that the common good is maximised, hence they describe their scheme as a *co-operative market strategy*.

All SSPs contain a number of pools of tokens, one for each SCP and service class pairing. Each time an SSP sends a service request to a SCP, one token is removed from the relevant pool. An empty pool indicates that the associated SCP cannot accept more requests of that type from the SSP. Tokens are periodically assigned to pools by a central auction algorithm which calculates appropriate token allocations based on bids from all SSPs and SCPs in the network. SCP bids consist of unclaimed processing capacity for the coming control interval and the processing requirements of each service class. In a similar manner, SSP bids consist of the number of expected service requests for each service class over the next control interval. These estimates are simply set as the number of arrivals in the previous interval.

The objective of the auction process is to maximise expected network profit over the next control interval. To do this, the auction maximises the increase in expected 'marginal utility', measured as the ratio of 'marginal gain' and 'marginal costs'. The expected marginal gain associated with allocating an additional token to a SSP is defined as the profit associated with consuming it times the probability that it will be consumed over the next interval. The expected marginal cost associated with issuing a token from a SCP is defined as the ratio of the processing time consumed and the remaining processing time. In this manner, tokens will typically be allocated to SSPs with higher bids, i.e. those expecting greater numbers of

requests for higher profit services over the control interval. The net effect of the auction process is that tokens are allocated in a manner that balances the arriving traffic load across all SCPs, subject to maximising the overall network profit.

2.5.3. Performance Control in TINA

There has been a considerable amount of research published on TINA networks in general but relatively little of it relates to performance issues. Parhar & Rumsewicz [1995] have done some initial investigations of performance issues in TINA. Sperryn *et al.* [2000] present a technique to assess performance metrics for objects executing in the TINA DPE. Kihl *et al.* [1998, 1999] and Widell *et al.* [1999] have investigated feasible load balancing algorithms and overload control mechanisms for TINA, and study how the distribution of computational objects affects the performance of the TINA network. Kihl *et al.* [1997] have identified the impact of Computational Object (CO) placement on performance in TINA networks through simulations. They conclude that network performance is highly sensitive to how the COs are distributed among network nodes. Choo *et al.* [2002] review the area but mainly reference the work of Kihl. However, they report on how this work has been implemented in the SATINA [Sperrin, 2000] trial platform.

We review Kihl's work here ([Kihl *et al.*, 1997, 1998, 1999], [Widell, Kihl *et al.*, 1999]) as it is the only substantial work, as far as we are aware, detailing TINA performance control mechanisms and their evaluation and is directly related to work undertaken in this thesis.

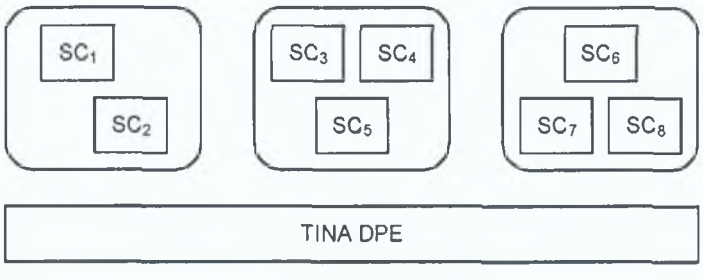


Figure 2.12: Model of TINA Service Components

In [Kihl *et al.*, 1997, 1999] and [Widell, Kihl *et al.*, 1999], the authors consider a model of a set of TINA processing nodes with each node hosting a set of communicating TINA Service Components (SCs). The DPE below the SC layer provides communication and location transparency between SCs. Thus the same SCs can be placed on several nodes. However, SCs are not permitted to migrate between nodes. Due to the multiple instances of a SC on different nodes, it is feasible to apply load-balancing algorithms in the TINA network to improve the throughput and delay during heavy traffic. If a particular node suffers from heavy traffic, the other nodes can relieve the situation by sending traffic elsewhere in accordance with some load balancing decision. Five different load-balancing algorithms are investigated in their

papers These algorithms use different load detection metrics at each node to decide a course of action, with measurements being evaluated and decisions made at the end of discrete control intervals The algorithms proposed are

Random A SC instance is chosen randomly and fairly from the set of nodes hosting that particular SC type

Shortest Queue The SC instance is chosen from the node with the shortest processor queue

Acceptance Probabilities Two metrics are used for each node $N_{tot}(i)$, the number of messages sent to node i and $N_{rej}(i)$, the number of signals sent to node i that have been rejected The estimated acceptance probability of node i is then given as

$$A(i) = \left(N_{tot}(i) - N_{rej}(i)\right) / N_{tot}(i)$$

Acceptance probabilities are calculated for each node and an SC instance is then chosen on node i with probability

$$P(i) = A(i) / \sum_{\forall i \in V} A(i)$$ where V is the set of nodes that hosts the particular SC

Load Status Values Each node uses a metric *L(i)* which denotes *load status* of node i *L(i)* is decreased by one when there are any messages rejected on node i, by the end of a control interval, otherwise it is increased by one An SC instance is chosen on node i with the probability

$$P(i) = L(i) / \sum_{\forall i \in V} L(i)$$ where V is the set of nodes that hosts the particular SC

Ant Based Special objects, ants, on all nodes make queries to surrounding nodes at random intervals The round trip times for the queries for each node i are collected and a corresponding weighing *W(i)* calculated *W(i)* is derived from the reciprocal of the round trip times and taken as a measure of load That is, the assumption is made that the load on a target node is inversely proportional to the ant round trip time for that node The routing probability for node i is then calculated as

$$P(i) = W(i) / \sum_{\forall i \in V} W(i)$$ where V is the set of nodes that hosts the particular SC

The authors investigate the performance of these algorithms in a simulation environment The simulation model consists of 10 processing nodes hosting 5 different SCs Communication patterns between SCs are based on a simple TINA service representing the equivalent of an ordinary telephone call Execution times of SCs are chosen more or less arbitrarily DPE

costs, for SCs communicating across the network, are accounted for by multiplying the SCs' execution times by 5 (again an arbitrarily chosen value) Transmission times in the network are modelled as extra execution time in the sending and receiving node New arrivals to the network are modelled as Poisson streams and evenly distributed amongst object instances representing the system users Each node is modelled as a single server system with infinite First In First Out (FIFO) job queues They investigate a low traffic and a high traffic scenario Two different SC allocations in the network are examined The first, *balanced*, has SCs distributed between their hosts in an even manner The second, *focused*, has a majority of SCs concentrated on a few nodes Their results show that for all algorithms high load in the *focused* scenario performs worst in terms of throughput with *Shortest Queue* and *Load Status* performing better than the others in this scenario

The authors also make the distinction between *internal* and *external* performance controls in TINA networks *Internal Overload Control* has the objective of protecting each node individually in the network from overload *External Overload Control* has the objective of optimising the overall network performance When an overload condition is detected within the network, new requests are rejected at the ingress point to the network That is, service set-up messages are rejected before they can enter an overloaded network

The above load control schemes may be categorised as internal overload controls With external overload control in mind, Kihl *et al* [1998] derived a simple control that rejects new calls at the ingress point to the network when rejections are detected internally Also, further to their previous observations on the criticality of proper assignment of SCs to network nodes, they derive a Mixed Binary Integer programming problem and apply it to finding the optimal distribution of computational objects in a TINA network The objective is to maximise the overall network throughput whilst maintaining processor load at or below a given level

A network of N fully connected nodes is considered There are M Computational Object (CO) types, all of which are required to execute a single service The arrival rate of new service requests to the network, λ, is to be maximised, with the constraint that load on each node must remain below a level of ρ_t The binary decision variable y_{mn} is defined such that it is equal to one if object type m is located on node n and is zero otherwise Objects may be duplicated across any number of nodes and the total count of an object's copies in the network is given as

$$b_m = \sum_{n=1}^{N} y_{mn}$$

which gives one of the problem constraints It is assumed that an object has an associated processing load of x_m during execution of a service session and it is also assumed that load is

shared equally among all copies of a component, i e the load caused by component m on a node is equal to $\lambda\, x_m / b_m$ This gives the processing constraint

$$\sum_{m=1}^{M} y_{mn}\, x_m / b_m \leq \rho_t T \quad \text{where } T = 1/\lambda$$

The objective function is then to maximise T, the system throughput The resulting solution values of the decision variables y_{mn} give the placement of COs in the network It is only when y_{mn} is equal to one that a copy of CO type m is placed on node n

2 5 3 1 Comments on TINA Performance Approaches

We see considerable commonality between interests displayed in work done on TINA performance and our focus on performance of inter-working between Intelligent Networks and distributed service platforms We see the areas of common interests as

(i) Internal Performance Control In this thesis, we base our model of a distributed service platform on TINA computational objects Thus the objectives of internal load controls in our scenario are akin to those in TINA However, we note that the simple algorithms investigated by Kihl *et al* aim to protect individual nodes and are not globally optimal There is no direct load detection metric considered by the control, rather it is estimated indirectly only from locally available knowledge (with the exception of the queue length algorithm) Of course, this method has an advantage in that no load status related traffic is required in the network (except in the case of the ant based strategy, which is deemed to be very small in terms of traffic volumes) However, this will tend to make the controls less effective as the controller only has an approximate estimate of the load situation in the network on which to base decisions We propose that a globally optimal strategy with a more direct control over network performance parameters would be desirable

(ii) External Performance Control The need for an external control for TINA networks has also been recognised and Kihl [1998] has shown the benefits in terms of network throughput However, as we have stated earlier, it is desirable that the *internal* performance controls and *external* performance controls (at the gateway in our case) should be integrated and co-ordinated in an optimal way to provide the most benefit to the service user and network operator TINA performance work has as yet not taken this approach

(iii) Placement of Computational Objects As investigated by Kihl *et al* , objects need correct assignment to network processors in order to guarantee high network throughputs However, we note that the model presented in [Kihl, 1998] does not account for the cost of remote communications, which is a major performance factor in middleware systems Also, the model assumes that all component copies process the same amount of requests This

assumption restricts possible solutions considered and, in general, will not give an optimal solution. Also other aspects such as differentiation between service types, and thence fairness to user types and profit optimality, have not been considered. We contend that the optimisation model should include these aspects as they have been of considerable importance in Intelligent Network load control ([Jennings, 1999] and [Lodge, 2000]).

In general we subscribe to these approaches for TINA performance, namely the importance of optimal component placement and of internal and external controls, however, we believe that better solutions are required, based on the following premise. If load can be balanced optimally internally, then rejection of messages should only be necessary at the ingress points to the network and not internally. This assumes that delays may be kept at low enough values so that users do not abort service sessions and that users do not abort for other (non delay related) reasons. Further, if an optimised external control operates to only reject the initial service setup requests at the ingress points, there will be no 'wastage' due to partially completed service sessions being prematurely ended (due to message rejection by the external control). The internal and external controls must also be optimally co-ordinated so that the external control functions to reliably maintain the loading in the network below some given threshold.

Optimisation of distributed software has been much researched in the wider field of distributed systems performance. To progress our investigations we now review this area.

2.6. Distributed Systems Performance

There is already a large volume of research in the performance of general distributed systems, much of which is also relevant to distributed telecommunication service networks. We review the literature here and examine issues relating to the telecommunication services domain and in particular identify the salient issues relating to work undertaken in this thesis.

2.6.1. Introduction to Performance Models of Distributed Systems

Generally, Distributed Computing Systems are complex and display a range of properties associated with parallel processing, that are not associated with sequential processing. This makes control and analysis of their performance more difficult. Unlike sequential processing systems, distributed systems pose the problems of design of the parallel algorithm for the application, partitioning of the application into tasks, co-ordinating communication and synchronisation, and scheduling of the tasks onto the machines. Given these complexities, achieving performance improvements or optimisations in such systems, by controlling system design and execution, is a difficult problem. To formulate useful models that are also tractable, we need to capture the salient features of the overall system whilst assuming certain

levels of abstraction in terms of the constituent components and the behaviours of the system. Here we discuss these components and behaviours along with the simplifying assumptions generally made in modelling.

In a distributed system model, the execution system is assumed to be a network of processing nodes each with its own local memory unit so that processing nodes do not share memory and communication relies solely on message-passing over a network. The main elements and behaviours of the general systems model, and how they may impact on overall distributed system performance, are:

Processing Nodes: A processing node is a collection of physical devices including CPUs, buses, memory units, storage devices and network interface devices. Processing node performance will have an obvious impact on overall system performance. In particular the number, structure and speeds of processors on each processing node, amount of available memory, OS operation and general machine architecture will impact on the effective processing power available to a distributed application. For the purposes of constructing a manageable distributed systems models, the details of the interaction of these internal physical devices are normally ignored. Usually of primary concern in modelling are the overall processing speeds of processing nodes and the behaviour of message queues at the nodes.

The processor speed determines the times required to execute modules of a parallel program and ultimately impacts on the overall system response time and throughput. The probability distribution of this processor service rate may be considered in the model or simply the mean service rate may be considered.

How message queues behave will also impact on performance. It is normally assumed that messages arriving for processing at a node may be added to a buffer if the processor is busy serving another message and that queued messages are then served according to some strict discipline, such as first in first out (FIFO). As buffers consume system memory, there may be a limit set in the model on the number of messages in the buffer or on the total memory consumed by queued messages. It can be appreciated that the average and maximum length of queues and the queuing disciplines impact on overall system performance.

The behaviour of each processing node in the system may be considered to be identical in terms of performance and capabilities (a homogeneous system) or nodes may be modelled as having differing performance levels and capabilities (a heterogeneous system).

The Network: Processing nodes are connected by an interconnection network. Performance of the network is dependent on transmission times or bandwidth, physical structure and topology, routing policies and data and network protocol behaviours. As distributed

applications need to communicate across a network during their execution, the performance properties of the network will be a determining factor in overall system performance The details of the network are not generally considered in the broader distributed systems model Normally the network behaviour is considered in simple terms, for example, a constant time delay for message transmission over a network hop is assumed to be incurred The network topology may also be considered in determining the number of hops, and thus the overall transmission time, for messages between processing nodes

Remote Procedure Calls and Middleware A remote procedure call (RPC) mechanism provides encoding and decoding of messages between processes executing on different processing nodes In a middleware environment, such as CORBA, invocation of an RPC is provided by means of local procedure calls on a set of *stub* processes on the local machine On the remote machine a similar set of processes receives and decodes RPC messages, which are then presented to an application as a local procedure call From the applications perspective the call resembles an ordinary local procedure call, however, some additional processing on both the local and remote machines is incurred due to encoding and decoding of messages for transport over the network This additional processing can cause significant performance degradation in the system when a large number of messages are passed during application execution

The behaviour of RPCs is influenced by the particular middleware design Client-server systems, such as CORBA, exhibit certain behaviours depending on modes of operation selected by the application designer Blocking type calls halt execution of the client process until a return is received from the server Non-blocking calls do not wait for a server response before resuming execution These resource contention behaviours may or may not have a large impact on overall performance depending on the operation of the internal parallelisms at a processing node For example, threading all blocking client calls effectively renders the calls as non-blocking, as duplicate client threads may execute in parallel to blocked threads Additionally, server processes may be multi-threaded so that blocking calls can be executed in parallel at the server Of course, threads have overheads, which must also be considered

The exact client-server behaviour will depend on the particular middleware product and OS details and detailed behaviours may or may not be included in a performance model It is often assumed, for example, that client-server calls may be modelled as purely non-blocking, as high performance systems normally try to avoid blocking behaviour, and thus the behaviour can be assumed to be equivalent to a simple message passing scheme with queuing of messages at the server-side Other research has included client-server paradigms in detailed analytic models, such as Layered Queuing Network modelling, discussed in §3 1 9

The Distributed Application A distributed application is a collection of units of work which are required to be executed on processors in the network in order to perform some overall function Certain work units of the same application may be executed on different processing nodes in parallel in order to gain an overall performance advantage The way in which such an application is decomposed into work units, how their execution is shared amongst processing nodes and the degree of communication between work units during application execution are central issues for distributed systems performance Regarding these units of work, we further qualify their assumed behaviour and make a distinction between *jobs*, *tasks* and *distributed objects*

Jobs Jobs are indivisible units of work, that is, their execution cannot be divided among processors Different jobs are considered independent of each other, that is, there are no structured execution relationships between them other than possible sharing of processing time and processor queues For example, a simple distributed application execution may be modelled by a collection of such jobs were all jobs must be executed but in no particular order and with no data sharing required between jobs A collection of jobs may also model a number of different independent applications executing in parallel on the same set of processing nodes

Tasks Like jobs, tasks are units of work that are atomic in that their execution may not be pre-empted nor distributed across processing nodes They normally model small units of logic that perform operations when executed in the system, for example a task may represent the processing required on receipt of a remote procedure call Unlike jobs, sets of tasks relating to the same application normally have a data and precedence dependency i e the tasks may need to share data to complete execution of a distributed application and may need to execute in a particular order In practice sharing of data and execution control is effected by means of RPCs between processes executing different tasks A distributed application may be modelled as a collection of interacting tasks It is normally assumed that the communication pattern and data dependencies between tasks during execution of an application are well defined (for example by means of class diagrams and related Message Sequence Charts (MSCs))

Distributed Objects In a middleware architecture, such as CORBA, distributed objects are defined in terms of an interface which describes a collection of method calls on the object Effectively this notion of a distributed object, from a performance modelling point of view, implies a collection of related tasks (the object's method calls) which are not distributable (i e must all reside at the node where the distributed object is instantiated) Distributed objects of an application are normally interdependent in that an object's tasks are executed in relation to the execution of the tasks of other distributed objects

Partitioning and Granularity of Tasks: How an application is partitioned into distributable tasks will have a large impact on achievable system performance. Generally speaking, partitioning into many fine-grained tasks will increase the ability to take advantage of execution concurrency across the network. However, this advantage may be offset by increased communication requirements between tasks. Partitioning into large sized tasks may reduce communication requirements but also reduces flexibility in task distribution and parallelism. Given a particular partition of an application, the actual system performance realised is determined by the distributed scheduling scheme.

Distributed Scheduling: Scheduling is concerned with where and/or when tasks or jobs should be executed on network nodes. Scheduling is an important issue as a poorly designed schedule causes inefficient use of processing and memory resources and introduces software and hardware bottlenecks. Subsequently the system performance can deteriorate. Scheduling is a large class of complex problems that encompasses related sub-problems such as load balancing, load sharing and task and job allocation.

General Task Scheduling: This is the ordering and allocation of tasks, communication and data to processors. The schedule normally has an application centric performance objective such as minimising application execution time and is performed once at design time.

Mapping or Task Allocation: Assignment of tasks to processors to gain a performance advantage without regard for order of execution amongst tasks. This is a sub-problem of general task scheduling but is normally driven by a system centric performance measure such as minimal processor utilisation, minimal communications, or maximal system throughput.

Load Sharing: Load sharing is similar to task allocation but normally involves assigning jobs, rather that inter-dependent tasks, to processors at run-time with the aim of achieving some dynamic performance goals, such as equalising queue lengths in the system.

Overload Control: The goal of overload control is to prevent system performance from degrading in an uncontrolled fashion under heavy load. As a system's load increases towards saturation, response times typically grow very large. Under such conditions, it is often desirable to shed load in some controlled manner rather than cause all users to experience unacceptable response times. *Admission Control* is a specific form of overload control where a proportion of traffic is rejected at the entry point of the system when some performance threshold is exceeded. Traditionally, overload control is a telecoms domain concept employed to guarantee a certain level of Quality of Service (QoS) to users but it has rarely been seen implemented in generic distributed systems. However, we consider it an important factor in providing high performance distributed systems for telecom service execution and include it here as a desirable distributed systems behaviour.

Application Users and Workload Characteristics Application users may be considered as part of the distributed system behaviour as they generate demand for application execution and may interact with the application during its execution Service requests may be in the form of simple independent jobs or require a set of related tasks to be executed There may be a number of different application types in the system, which require different job types or task sets to be executed Users that execute different jobs or task sets are referred to as belonging to different *customer classes*

The stochastics of service request arrivals and user interaction periods can impact on distributed application performance As mean service request arrival rates increase it is expected that average service times in the system will increase due to increased queue lengths in the system Generally, service request arrivals with large inter-arrival variance or arrivals that are bursty in nature will produce longer delays in the system Often simple stochastic models are assumed when modelling random externally driven events such as arrivals For example, the Poisson process has been used extensively to model service requests from large populations of independent users

2.6 2 Performance Metrics for Distributed Systems

Performance metrics for distributed systems are required in order to assess system performance and to provide goals for performance optimisation measures, such as scheduling There are several in common use

- **Speed Up** This is a measure of parallelism efficiency and may be taken as the ratios of the execution times in a single processor system and in a system of multiple similar processing nodes This gives an indication of how a network scales as the number of processors is increased Ideally, speed up should increase in proportion to increase in the number of processing nodes in a distributed system
- **Communication to Computation Ratio** of a parallel program is defined as the ratio of the average communication time and average processing time for an application This is similar to speed up and indicates the efficiency of a given application distribution scheme
- **Processor Load** The fraction of time a processor is busy processing tasks Generally of most concern in distributed systems is a load sharing metric with gives a measure of load imbalance across processors This may be stated simply as the difference of maximum and minimum loads This is a useful measure as often load imbalances are more an indicator of expected performance than the average system load
- **Make-Span** This is the total application run time in the system It does not account for queuing delays but does include time that tasks are blocked waiting for other tasks to execute If there is no blocking involved, then make-span is simply computed as the total execution time of all application tasks

- **Throughput** We define throughput as the number of application sessions, completed per second When there are no losses in the network, due to lost messages during congestion or overload conditions, the throughput will be equal to the offered traffic intensity This metric is most useful when assessed in conjunction with system load or system delay Often of interest is the throughput achievable for a given maximum load or delay
- **Round Trip Delay** This measure gives the average time for completion of an application session It thus includes all task processing times and waiting times due to queuing of messages in the system It is an important metric as it determines the Quality of Service that may be offered to users in terms of responsiveness of a system
- **Queue Length** Queue lengths in the system are sometimes used as an indirect indication of load and delay in a system

2 6 3 Optimising Distributed Systems Performance

The performance of a distributed system is largely determined by the available resources and technologies (processing nodes, network infrastructure, operating systems, middleware) and on the design and deployment of the distributed software and data Given that the available physical resources and technologies are generally fixed due to cost or technical constraints, the distributed application design, particularly in relation to optimisation of partitioning and scheduling, offers the main opportunity for improving the system's performance Even when the amount of physical resources are not constrained, it is largely the efficiency of the partitioning and scheduling schemes that determine whether or not a performance gain commensurate to expenditure on resources can be achieved Indeed, until an application is partitioned and scheduled onto a network, it may be difficult to predict how much processing power and network bandwidth is required to achieve a required system performance

Partitioning of applications for distribution has a strong effect on possible system performance Partitioning may be optimised by deciding the appropriate level of granularity for a distributed application The granularity at which an application is divisible impacts on the potential for improving the performance of its distribution as the number of potential distributions is inversely related to the distribution granularity If the number of distributions is insufficient, none may offer good performance However, if the granularity is too small, the tasks of partitioning an application and realising the distribution may become prohibitively expensive Optimised partitioning schemes based on minimising communications costs have been investigated in Purao *et al* [2002] Systems have been developed, for example COIGN [Hunt & Scott, 1999], which automatically partition applications at compile-time, but use of such methods is not common practice

Although partitioning and scheduling impact on each other and thus should ideally be considered together, this is not generally the case and scheduling is normally treated

separately having first decided on an appropriate partition of the application Indeed, in the case of telecom service applications, systems are generally based on prescribed sets of service components (for example TINA ODL interfaces) which may restrict or predetermine application partitioning into distributable objects For these reasons, in this thesis we solely focus on optimisation of *scheduling* of distributed systems and do not consider partitioning The next section reviews in detail the area of distributed system scheduling

2.6.4. Scheduling in Distributed Systems

This thesis is primarily concerned with issues relating to *task allocation* and *load sharing* which are problem areas in the wider field of *scheduling* We give a brief overview of the general area and then focus on scheduling in distributed systems, our main concern The computational complexity of scheduling problem solutions is an important practical issue and is also considered here

There is a large body of literature relating to scheduling problems, extending over a long period of time and applying to many application areas Unsurprisingly, the terminology in the literature is variable A number of unified taxonomies for scheduling algorithms have been proposed (Casavant & Kuhl [1988], Wang & Morris [1985], amongst others) but there still remains an overall variance in terminology We do not try to align the differences here, but have chosen a terminology sufficient for our discussions

The general scheduling problem may be described as that of optimally assigning a set of tasks to a set of resources given the execution costs of tasks and the execution precedence dependencies between tasks The objective of a solution is normally a performance related goal, such as that of minimising the average time required to process tasks There are two aspects to scheduling *allocation* and *sequencing* Allocation may be considered as answering the question Where should tasks be executed? Sequencing answers When should tasks be processed in relation to the processing of other tasks Both allocation and sequencing may be unified in the same scheduling problem, or they may be considered separately

In computer systems, scheduling problems can be broadly categorised according to the nature of the processing system being scheduled A distinction can be made between scheduling of a single processor and scheduling of multi-processor and distributed processing systems On a single processor, scheduling is concerned with the assignment of processor time-slices to tasks (processes) which are waiting to execute This is refereed to as *local scheduling* In contrast, multi-processor and distributed systems are concerned with *global scheduling* which decides the allocation of entire tasks to different processors in order to achieve performance goals for the system as a whole Global scheduling thus considers the course grain properties of the system (e g processor speeds and network topology) and the properties of the tasks

being scheduled (e g mean processing time required for each task and inter-task communication costs) Global and local scheduling concerns are normally considered to be separable and their interaction is generally not considered

There are also differences in approach between global scheduling in multi-processor and in distributed systems and these can be attributed to the nature of the coupling between processors in the system Multi-processor systems are generally considered to be tightly coupled as they have an efficient communications mechanism, such as a shared memory Distributed systems provide communication between processing nodes via message-passing over a network and are considered loosely coupled Tightly coupled systems can synchronise the parallel execution of tasks on their processors and scheduling normally considers task execution sequencing and timing, as well as allocation, to gain further performance advantages Loosely coupled systems cannot accurately or practically synchronise the timing of tasks due to variable communications latencies introduced by the network and the problem of scheduling is generally restricted to that of allocation alone

The distinctions, arising from the nature of multi-processor and distributed systems, divide global scheduling into two quite disparate areas of study that of allocation with sequencing (often refereed to simply as scheduling) and that of allocation alone Each area has its own set of related performance goals, system assumptions and solution methodologies There is a large body of research pertaining to allocation with sequencing for multi-processors and a literature review may be found in [Baumgartner & Wah, 1990] As we are primarily concerned with performance of distributed systems, and not with tightly coupled systems, we direct our attention towards problems of allocation alone

2 6 4 1 Task Allocation

Task allocation is simply the choice of a mapping of a set of tasks to a set of processors so as to achieve some pre-defined performance goal This goal is usually represented as some objective function that may include a combination of several criteria such as equal load sharing between processors, maximisation of the degree of parallelism, minimisation of the amount of communications between processors, etc Several different aspects of the distributed system may be represented in the problem task execution times, amount of inter-task communication, topology of communications network, processor capacities, allowable processor load skew, etc In task allocation problems, these parameters are considered to be deterministic and known *a priori* This type of scheduling is thus termed *deterministic* and *static* That is, the schedule is determined at design-time and only considers expected values for task processing times and arrival rates to the system We next describe the static task allocation problem in detail and review solution methods from the literature

Static Task Allocation

To give an insight into the static task allocation problem, we consider a representation of a generic distributed system depicted in Figure 2 13 The key elements in the system are a set of tasks to be processed, a set of processors which communicate over a network and a scheduler which allocates tasks to processors The tasks are considered to be dependent, in that they act together to perform some overall service and, in order to co-ordinate their execution, they must communicate with each other This relationship is usually expressed as an undirected connected graph where nodes represent tasks and edges represent communication dependencies between them Note that the precedence of execution of tasks is not considered in the task allocation problem

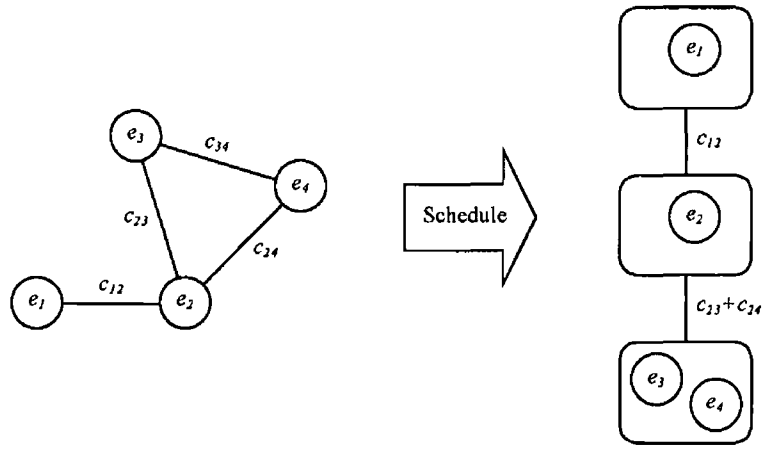


Figure 2 13 A Task Allocation Schedule *An undirected task graph (left) specifies execution times for tasks and remote communications costs for task pairs The scheduler assigns tasks to processors (right)*

It is assumed in the model that tasks allocated to the same processor do not incur any inter-task communications costs When tasks are assigned to different processors, substantial costs, associated with remote procedure call overhead and network latency, are incurred The system model may also stipulate that each task has a different execution time on each processor and that there are limits on processing and link capacities The goal of the scheduler is then to make an allocation of tasks to processors such that some performance measure (derived from the communications costs and execution costs in the system) is optimised As the scheduler is assumed to have knowledge of task execution times and communications costs *a priori*, this problem is described as *static task allocation*

There can be a number of competing objectives that will affect overall performance of the system Intuitively, to minimise the communications costs in the network, all tasks could be allocated to one processor so that there is no inter-processor communication We could further stipulate that the processor that gives the lowest total execution time over all tasks

should be chosen Of course, this approach neglects the performance advantages of processing tasks in parallel on different processors so that overall throughput can be increased However, allocating tasks to different processors will also increase communications costs that will in turn cause some decrease in throughput Other constraints, such as limited link capacities and balancing of load, may also need to be included in the problem The scheduling decision thus becomes a balancing act between a number of competing goals

Several approaches have been taken to formulating and solving static task allocation problems The main approaches may be categorised as graph theoretic, mathematical programming, heuristics and approximation approaches, although there is some crossover between these categories in many of the solutions proposed in the literature Given that allocation problems are generally NP-Hard combinatorial optimisation problems [Papadimitriou & Steiglitz, 1982] and thus are unlikely to have polynomial-time solutions, algorithms often aim for sub-optimal, but efficient, solutions when the problem size is large Approximation approaches may be used to achieve fast sub-optimal solution methods with known solution accuracy Heuristic approaches can also provide fast sub-optimal algorithms but generally do not give any guarantees on solution accuracy Graph theoretic and mathematical programming approaches can also be used in combination with heuristics or approximation For problems of sufficiently small size, exact solution methods can achieve an optimal solution in reasonable (low order exponential) time Both exact graph theoretic and mathematical programming approaches have been employed in optimal solution methods We review the various approaches below

Graph Theoretic Approaches

The birth of the graph theoretic approach to task allocation may be attributed to Stone *et al*, [1977] who use the *max flow-min cut* theorem from graph theory [Diestel, 1997] to search for an optimal allocation of tasks to processors Stone's objective for the problem is to minimise the sum of execution and communications costs

An undirected connected graph is constructed where nodes represent both tasks and processors (Figure 2 14) An edge between two tasks is weighted with the corresponding inter-task communication cost For instance, this cost may be taken as the volume of data exchanged between two tasks during execution of the task graph An edge between a task and a processor is weighted with the execution time of the task on the *other* processor A potential assignment of tasks to processors is given by a cut of the graph where the cut creates two disjoint subsets with P1 and P2 in different subsets The sum of the edge weights crossing the cut gives the total execution and communications costs for the allocation Thus the problem is to find the minimum cost cut This may be found in polynomial time for two processors by

application of the max-flow min-cut theorem and the Ford-Fulkerson max-flow algorithm (see [Papadimitriou & Steiglitz, 1982]) This formulation may be extended to an n-processor system However, Stone notes that the solution involves application of an n-dimensional min-cut algorithm, which becomes computationally intractable for even moderately large n

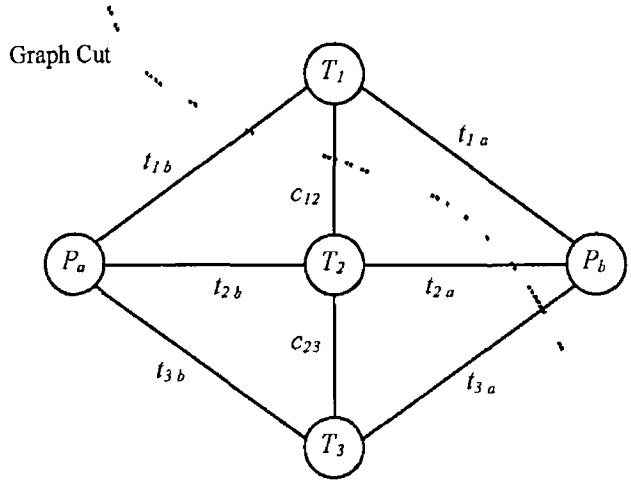


Figure 2 14 Stone's Graph Cutting Method

Shen and Tsai [1985] propose a more efficient *graph matching* approach to the n-processor problem They formulate the problem as two separate graphs representing the set of tasks and the set of processors in the system Sets of weights on the task graph nodes represent task execution times at each processor Edge weights in the task graph represent communications costs between tasks In the processor graph, edges represent connectivity between processors Each graph match then corresponds to a possible task allocation An A* Search method [Nilson, 1971] is used to reduce the possible number of matchings considered and an optimal solution is found with relative efficiency up to about n=20 Lo [1988] proposes a sub-optimal efficient heuristic method for Stone's n-processor problem A "grab" phase first produces a partial optimal assignment by employing a mapping from the n-processor system to a set of two-processor systems to which Stone's solution method is applied Two further sub-optimal 'greedy' phases assign the remaining tasks with the goal of minimising communications Ramakrishnan *et al* , [1993] solve the problem using a combination of A* Search and Lo's "grab" method to achieve an optimal solution method with better efficiency than Shen and Tsai's algorithm Kfil & Ahmad [1998] also propose a similar A* Search graph matching algorithm which is suitable for execution on a parallel machine and report the added benefit of reduction of the algorithm's memory requirements

Mathematical Programming Approaches

Several mathematical programming approaches have also been proposed to finding an optimal solution to the task allocation problem (Refer to §3 3 for a review of mathematical

programming) The classical formulation is proposed by Chu *et al* [1980] who uses a 0-1 programming approach The set of m tasks is to be assigned to n processors Processing costs are defined as the matrix $E=\{e_{ij}\}$, $i=1, m$, $j=1, n$ where e_{ij} represents the execution cost of task i on processor j Communication costs are defined by the matrix $C=\{c_{ik}\}$, $i,k=1 m$, where c_{ik} represents the communication cost incurred when task i and task k are assigned to different processors Communication cost is zero if tasks are assigned to the same processor The binary decision variable x_{ij} is defined and is equal to one when task i is assigned to processor j and zero otherwise The processing and link capacities are not constrained As with Stone's problem, the objective is to minimise execution and communications costs Total execution cost is given as

$$\sum_{i=1}^{m}\sum_{j=1}^{n} e_{ij}x_{ij}$$

And the total communication cost is given as

$$\sum_{i=1}^{m-1}\sum_{j=1}^{n}\sum_{k=i+1}^{m} c_{ik}(1-x_{ij})x_{kj} \quad = \quad \sum_{i=1}^{m-1}\sum_{k=i+1}^{m} c_{ik}\left(\sum_{j=1}^{n} x_{kj}\right) - \sum_{i=1}^{m-1}\sum_{j=1}^{n}\sum_{k=i+1}^{m} c_{ik}x_{ij}x_{kj}$$

Since the sum in parentheses is one and the summation of the remaining c_{ik} is a constant, the first group of terms may be removed from the objective function Thus adding execution and communication costs the minimisation problem is

$$\text{minimise} \sum_{i=1}^{m}\sum_{j=1}^{n} e_{ij}x_{ij} - \sum_{i=1}^{m-1}\sum_{j=1}^{n}\sum_{k=i+1}^{m} c_{ik}x_{ij}x_{kj}$$

Subject to the constraints

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall\, i = 1, \ldots, m$$

which stipulate that each module be assigned to exactly one processor Chu notes that this general problem has the form of a quadratic binary programming problem which may be linearised, to a binary integer linear program (BILP), by a change of variable and addition of appropriate constraints The BILP may then be solved with standard techniques for integer linear programming such as a branch and bound method

Billionet *et al* [1992] consider the same problem as Chu and propose a more efficient solution technique They consider the quadratic form of the cost function (as above) and note that without constraints, and by relaxing the decision variables to be real, the problem is

efficiently solvable by considering it as a max-flow problem in a bipartite graph (see [Papadimitriou & Steiglitz, 1982]) The authors then construct a branch and bound method to solve the original problem, which works on a Lagrangian relaxation of the original constraints They report an algorithm efficient enough for large problems (20 processors and 50 tasks) Recently, Lewis *et al* [2004] have shown how the same problem may be re-formulated as an unconstrained quadratic binary program (UQP) which has an exact solution method with even better computational properties

Bastarrica *et al* [1998] consider a similar problem where the objective is to minimise the overall communication cost That is, only the quadratic part of Chu's objective function is included They add storage and link constraints to the problem Each task is assumed to consume a given amount of storage on a node and the total storage on each node is constrained The total amount of communications between nodes is limited to a given link bandwidth value where the network is assumed to be fully connected so that any two nodes communicate via a single link Similarly to Chu, the authors linearise the objective function to form a BILP They employ a generic branch and bound integer linear program solver and discuss the complexity of an exact solution to their problem They note that current branch and bound solvers can run in near linear time in the number of integer variables allowing problems with over 1000 integer variables to be solved routinely under current (circa 1998) computing power

Heuristic Approaches

Most graph theoretic and mathematical programming approaches state the problem in such a way that it may be solved exactly, provided that the execution time of the algorithm is reasonable With larger problems in mind, researchers have proposed sub-optimal, efficient heuristic solutions to the task allocation problem

Efe [1982] proposes a heuristic algorithm whereby tasks are 'clustered' together in such a way that communication cost between the resulting clusters is minimised Whole clusters are then assigned to processors in an iterative manner in order to achieve load balance between processors Chu [1980] has proposed a similar approach whereby tasks are clustered until the number of clusters is equal to the number of processors whilst attempting to cluster tasks with heaviest communication together Tasks are then moved from processor to processor so that a load balance is achieved Senar *et al* [1998] propose an approach whereby task clustering is first used to contract the task graph The second stage takes the contracted graph and tries to successfully match it to the network of processors The objective is to minimise both processing and communications costs Bowen *et al* [1992] propose a hierarchical clustering and allocation method that aims for improvement on overall communication cost whilst satisfying upper and lower bounds on processor usage Sadayappen *et al* [1990] have also

proposed a clustering type solution based on the efficient Kernighan-Lin graph bisection heuristic [Kernighan & Lin, 1973].

More general random heuristic techniques have also been applied to this area. Simulated annealing has been applied to the task allocation problem by a number of researchers. Kazuo [2001] notes that although simulated annealing can perform effectively and avoids local optima traps in static task allocation problems, the standard method can take a long time to converge to a solution. They construct a standard simulated annealing model and then modify it by incorporating heuristics based on achieving load balancing and reduction of inter-processor communications. This method shows a speed up and they show that their algorithm has a solution close in accuracy to the standard simulated annealing method. Lee & Bic [1989] present evidence that, with a regular network topology, there are no significant local minima in the space of possible solutions. Hence, a faster form of simulated annealing called 'quenching' becomes appropriate for the task allocation problem. Quenching performs a rapid cooling schedule, rather than the normal slow cooling, which gives faster convergence to a minimum.

Genetic algorithms are another general random method that has been applied to task allocation. The idea of a genetic algorithm (GA) is to follow an evolution process based on operators such as mutation, inversion, selection and crossover. These operators are applied to find successively better local minima. The procedure continues evolving until it remains trapped in a local minimum. Singh & Youssef [1996] have formulating the task allocation problem in GA terms, and then evaluated various genetic algorithm parameters for obtaining best performance. Talbi & Muntean [1991] have proposed a GA that is suitable for implementation as a parallel program. They aim to solve for a task allocation that minimises a weighted sum of communication costs and the variance of load imbalance. Having optimised their algorithm parameters, they report good solution accuracy and performance when compared to hill climbing and simulated annealing. Park [1997] has applied a genetic mean-field annealing algorithm to the problem, which is a hybrid of GA and mean field annealing. They show that the hybrid algorithm combines the benefit of both methods and gives improved performance.

Other heuristic algorithms have been applied to the problem. Elsadek & Wells [1999] construct a heuristics model which aims to minimise inter-process communication time and balance processing load. They construct a greedy, locally optimised algorithm and also consider a randomised algorithm, which achieves more optimal solutions. Aguilar & Gelenbe [1997] propose a random neural network model for solving the problem and show comparable performance to standard genetic and simulated annealing algorithms.

In summary, the main difference in approach to solving static task allocation problems is in whether an exact or inexact (sub-optimal or approximate) solution is achieved. Although the problem is NP-hard, relatively efficient exact solution methods for reasonably large problems have been demonstrated using binary integer linear programming with branch and bound solution methods. Large scale problems have been tackled using heuristic approaches based on graph theoretic methods with heuristics, clustering heuristics, simulated annealing, genetic algorithms, neural networks and greedy algorithms. Exact graph theoretic methods (those that do not use heuristics) have not yielded efficient solutions. Apart from the solution methods employed, the approaches in the literature differ in terms of their objectives. Minimum communications costs, load balance and maximum loading have mainly been considered.

2.6.4.2. Dynamic Task Allocation

When all of the relevant system characteristics are known at compile-time, the allocation problems, like those discussed above, are known as *static task allocation* problems. In contrast, *dynamic task allocation* relates to task allocation (and re-allocation) during program execution that moves workload amongst processors in response to changes in system-state information, such as current request volumes and current processor loading.

Note that, unlike in static task allocation, dynamic task allocation implies that the processes executing tasks may migrate from node to node at run-time or that the same task may be available for execution on multiple nodes simultaneously thus allowing the movement or sharing of load. In static task allocation, each task is permanently assigned to a processing node. These additional complexities in the model fundamentally change the nature of the problem and make it more difficult to find optimal solutions. The majority of research pertaining to dependent tasks (as opposed to independent jobs) has concentrated on static allocation. Only recently has dynamic allocation been examined and then mainly in relation to process migration strategies. This recent interest may be accounted for by the recent proliferation of Distributed Object Systems, whose properties, such as location transparency, more easily allow relocation of objects in a network. Optimal distributed object allocation problems, which include the dynamics of task migration, are mentioned in §2.6.5.

2.6.4.3. Load Sharing

Load Sharing is a method of assigning jobs to processors with the aim of achieving some performance goals such as equalising queue lengths in the system or minimising queuing delays. This form of scheduling normally relates only to independent job models rather than systems of communicating tasks. *Load Balancing* strategies are a specific form of load sharing where the aim is to distribute the jobs in the system so that all processors perform an equivalent amount of work. Often the two terms are used interchangeably in the literature.

Load sharing schemes may consider probabilistic models of arrivals and service times to arrive at optimal scheduling decisions, whereas task allocation normally assumes only average values for task execution times From the perspective of the deterministic task allocation model, an increase in the frequency of execution of a set of tasks on the system processors will predict a linear increase in processor loading, for a given allocation of tasks to processors Choosing an optimal allocation is thus independent of user traffic volumes On the other hand, load sharing may consider the input traffic volume and the stochastics of the input traffic and the execution time of tasks In this case, the system performance metric (normally queuing delay or queue length) is non-linearly dependent on input traffic volumes, and traffic volumes become part of the problem definition This type of stochastic problem has been studied by a number of authors and the pertinent literature is reviewed below Note that this section references some queuing theory concepts The reader is referred to Chapter 3 of this thesis for a review of queuing theory

Tantawi and Towsley [1985] consider a network of connected heterogeneous processors, which may process any of a set of jobs arriving to the network All jobs are considered identical in terms of their processing requirements i e there is only one customer class in the system Any processor may receive a job and may subsequently choose to process it or pass it to another processor A job may only be passed off once and a communications delay is assumed when this occurs They consider the response time of a job in the system as consisting of a delay at the node due to queuing and processing and a delay due to any communication costs if jobs are passed It is assumed that the mean node delay is a function of the load of the node and that this function is increasing and convex It is also assumed that the network is a product-form queuing network, so that they may form a simple expression for the total mean response time of a job The goal of the load sharing strategy is then to find the transfer rates of jobs between nodes that minimises the mean response time (queuing, communications and execution times) of a job in the system There are no constraints placed on network links or processor loads They formulate and solve this problem as a non-linear, but convex, optimisation problem The output of the optimisation gives random splitting probabilities at each node for processing and forwarding in order to achieve the minimum response time

Similar forms of this stochastic static job scheduling have been studied by other authors Ross & Yao [1991] have considered an extension of this problem, where there is more that one job type in the system each with its own independent generally distributed execution time The service time distribution may vary with the job type and with the host processing the job Similarly to Tantawi and Towsley, they also require a convex and increasing delay function in order to achieve a solution Borst [1995] examines a similar problem accounting for different

customer classes He formulates the problem specifically as a network of M|G|1 nodes and finds the optimal assignment of customers to servers that minimises the mean total waiting time for customers Wolf & Yu [2001] consider a similar problem in the form of minimising overall response time in a Clustered Web Farm The web farm consists of a number of web servers and a number of web sites, which may be duplicated and distributed across the servers The web sites are computationally independent of each other and so can be considered jobs in our terminology They model queuing delay, by defining a response time function for each server This is an arbitrary function of overall traffic intensity at a server which may be obtained from simulation or measurement but is required to be differentiable, convex and increasing so as to achieve a tractable numerical solution method They formulate a non-linear optimisation problem with the objective of minimising the sum of all response functions over all servers whilst also maintaining server loads below a given value This is a separable convex resource allocation problem that they solve using methods based on Tantawi and Towsley's work Cardellini *et al* [1999] have applied a similar method for optimising web clients choosing between replicated hosts

Many forms of *dynamic job scheduling* have also been extensively discussed in the literature Generally dynamic schemes are based on relatively simple heuristics Some examples of well known schemes are Shortest Queue - an incoming job is assigned to the processor that currently has the least number of queue messages, Least loaded - the currently least loaded processor is assigned to the job Various other schemes have addressed migration of jobs between processors and have various objectives such as minimising queue lengths, minimising overall delay or balancing load Literature reviews may be found in [Bernardt *et al*, 1992] and [Yu *et al*, 1986]

2.6 5 Component Allocation

Classic scheduling problems focus on allocation of jobs or tasks to network processors With the more recent interest in Distributed Object Computing, scheduling mechanisms have been extended to deal with the allocation of distributed software components across a network The component model may be viewed as an extension of the task model and component allocation schemes in the literature have generally been based on earlier work on task allocation, particularly in relation to the graph theoretic and mathematical programming approaches We review the literature in this section in some detail as it is directly related to work undertaken in this thesis

For the purposes of our discussion here, we define a component as a software module that exposes interfaces allowing execution of its methods remotely A component may also be the source of remote method calls on other components We assume that a component is atomic

and may not be decomposed into smaller distributable objects For example, a component may represent a single CORBA object or a group of collocated CORBA objects Often in component allocation problems, method calls are not represented individually in the model but all communication between two components is amalgamated into a single flow or optimisation variable which represents some aspect of the total communication volume between two components The processing load associated with this communication flow between two components may also be the focus of the allocation problem Usually these flow problems may be represented as equivalent mathematical programming problems to obtain a solution

A number of authors have tackled the component allocation problem in different forms The scope of the problems differ in terms of the objective of the optimisation, the main optimisation variable chosen, whether or not multiple customer classes are considered, consideration of duplication of components, consideration of network topology and the solution methods employed

A comprehensive and quite general model of the component placement problem is proposed in [Anagnostou, 1998] The main focus is optimal placement of components in order to minimise communication costs in the network The problem is constructed as a linear mathematical programming problem with communication flow between distributed components as the main optimisation variable Their model takes into account the following aspect of the distributed system

- A set of communicating components that may be duplicated arbitrarily across processing nodes
- A network of processors An arbitrary network topology is allowed and a cost function associated with each network link is defined
- A set of service users of different customer classes with a set of associated demands they create for different service types Users are assumed to be fixed at certain nodes in the network

The model is composed of (i) $G(V,E)$ a network topology graph where V is the set of nodes and E the set of links, (ii) $G(C,F)$ a service graph where C is a the set of components and F the set of edges representing communication between them and (iii) Λ a set of edge labels associated with F which denote the traffic volume exchanged between components per unit traffic offered by a user to the network

The graph of all possible allocations of components to nodes is then constructed i e $G(\boldsymbol{C},\boldsymbol{F})$ where $\boldsymbol{C}=C\times V$ is the set of all components copied to all nodes and $\boldsymbol{F}$ is the set of all possible interactions between them

A set of constraints is then required to govern the communication flows inside $G(C, F)$ as follows If $(c_1\, c_2) \in C$ and $(c_2\, c_3) \in C$ are two pairs of components, with $\lambda' \in \Lambda$ and $\lambda'' \in \Lambda$ as their associated labels, then the sum of the traffic to/from all copies of c_1 to any copy of c_2 divided by the sum of traffic to/from all copies of c_3 to the same copy of c_2 should equal λ'/λ''

A second set of constraints is required to associate the flows in the network with traffic flows from users This is done by equating all flows involving the initiating components with the user demand flow Finally, the objective function is defined as minimising the product of the network flows and their corresponding distances over all possible edges, thus minimising total communications costs The authors illustrate how multiple customer classes may be accommodated by adding new service graphs to the formulation

It is noted by the author that it may be more reasonable to allow an inequality constraint to associate user demand and network flows so that the solution for the network flows need not meet all the demand from users (e g in an overload situation all user demand cannot be meet) This would be an important modification if a node capacity constraint (which is suggested as a possible addition) were incorporated into the model Otherwise, when the user demand exceeds a certain level there is no solution satisfying both constraints However, the equality constraint of network flows meeting user demand is the only constraint in the original problem which drives the solution away for the zero vector and the author notes that the use of the inequality constraint must be coupled with some addition to the cost function or an additional constraint to avoid the zero solution The author suggested that a cost is added to the objective which increases with the amount of unsatisfied user demand These issues are not fully addressed in the paper and we feel that a clearer and cleaner solution could be devised The authors also suggest a method to add component installation costs but state that this method would make the problem a hard combinatorial problem

Bastarrica *et al* [1998] also consider the problem of deploying software components in a network so that the overall remote communications cost is minimised The constraints considered are the available storage on each node and available bandwidth on links in the network It is assumed that each component is assigned to one and only one node The network is assumed to be fully connected so that any two nodes communicate via a single link and thus network routing costs are not considered Each pair of communicating components are assumed to generate a given amount of traffic and the bandwidth of links between nodes is constrained to given values Each component, when instantiated on a node, consumes a given amount of storage on that node The amount of storage is constrained on each node The authors formulate this problem as a mixed binary integer programming problem with the objective of minimising the total amount of communication in the network

The authors do not give an interpretation of their 'storage' variable but it can be seen that it may directly represent memory, disk space or, maybe more usefully, processing capacity This is a somewhat simpler model and more akin to a classic task allocation problem in that components are not allowed to be duplicated and the model is not driven by user traffic volumes from different customer classes

Kihl *et al* [1998] develop a simple mixed binary integer programming problem and apply it to finding the optimal distribution of Computational Objects in a TINA network The objective is to maximise overall network throughput whilst maintaining processor load at or below a given level

A network of N fully connected nodes is considered There are M computational object types all of which are required to execute a single service The arrival rate of new service requests to the network, λ, is to be maximised, with the constraint that load on each node must remain below a level of ρ_t The binary decision variable y_{mn} is defined such that it is equal to one if object type m is located on node n and is zero otherwise Objects may be duplicated across any number of nodes and the total count of an object's copies in the network is given as

$$b_m = \sum_{n=1}^{N} y_{mn}$$

This gives one of the problem constraints It is assumed that an object has an associated processing load of x_m during execution of a service session and it is also assumed that load is shared equally among all copies of a component, i e the load caused by component m on a node is equal to $\lambda\, x_m / b_m$ This gives the processing constraint as

$$\sum_{m=1}^{M} y_{mn}\, x_m / b_m \le \rho_t\, T \qquad \text{where } T = 1/\lambda$$

The objective function is then to maximise T, the system throughput

This approach differs from others in that load is the focus rather than communications costs The solution will tend towards load balancing across processors, however, the model does not account for the costs of remote communications This will generally tend to give too much distribution and non-optimal communication patterns between components All component copies are assumed to process the same amount of requests which restricts possible solutions considered and, in general, will not give an optimal solution

Avramopoulos and Anagnostou have considered the problem of optimal allocation of components to network nodes in the case where some of the components are fixed and some are *mobile agents* which may migrate from node to node as communication patterns change [Avram & Anag , 2002] In this work, the authors adopt a graph theoretic approach and seek

to minimise the network traffic that is incurred during component communications and during component migration The solution methods are formulated as a Mixed Binary Integer Programming problem The first problem they solve is that of locating a single copy of each component in the network in an optimal way The second problem considered is that of optimally migrating mobile components in order to reduce inter-component communication costs when communication patterns between components change

In the first problem, a network of nodes, denoted as the set V, and a set of interacting components C are considered A subset of these, C_x, are *fixed* components that are pre-assigned to certain network nodes and the remaining set of components C_m are considered *mobile* and are to be optimally assigned to nodes A graph $G(C,F)$ is then constructed where the set of nodes C represents the set of components and the graph edges F represent messages passed between components Labels $w_{\{c\,c\}}$ are assigned to edges to denote the volume of messages exchanged between components The cost of communication between components residing on different network nodes is considered to be the product of this traffic volume and distance between the corresponding nodes, denoted $d(v,v')$, where d is some distance function $d\ V \times V \rightarrow \Re$ The authors consider the problem of finding the assignment of mobile components C_m to the set of nodes V such that network traffic is minimal, that is minimisation of the cost function

$$C = \sum_{f \in F} \left[w_{\{c\,c\}}\ \ d(v,v') \right] D_f$$

where D_f is a binary decision variable, defined over all possible assignments of components to nodes, i e $C \times V$, which indicates whether or not two particular assigned components interact The following constraints are then added to the problem

- Only one copy of each component may be assigned in the network
- Only one pair of assigned components may have a non-zero flow between them and
- A component must be assigned a to node for there to be non-zero flows associated with it

The possible locations of mobile components are not constrained (although this is possible) There are no constraints relating to network bandwidth, processing capacity, etc

In the second problem, re-configuration of the component configuration is sought by means of migrating components from one node to another, when communication patterns between components change Migration is considered to occur only at the end of a *phase* During each phase the component communication pattern is constant and migration occurs in response to a change in anticipated communication patterns for the next phase Each mobile component is considered to have a size and an associated cost is incurred if it migrates to another node The

optimal routes for migration of components is sought, such that the overall cost of communication and the transportation costs are minimised Note that the first problem solution may be applied to the problem of finding an initial component configuration for components at design-time whose locations will thence remain fixed The second problem is only applicable to components that may migrate at run-time

The model does not consider different service classes, or limits on processor loading or network bandwidth Both problems are mixed binary integer programming problems The authors show that it is NP-Hard for N>2 Although a NP-Hard *design-time* problem may be tolerable, from a complexity point of view, it is uncertain if such a problem would be applicable to the dynamic run-time migration problem

Silaghi & Keleher [2001] consider a somewhat similar scenario to Avramopoulos & Anagnostou but apply simple heuristic decision policies, rather than optimal programming models, to achieve a solution They also consider a network where only one copy of a given component may be active at any time but this component instance is allowed to migrate from one processor to another The algorithm operates as follows the target processor of each message to be sent is evaluated as a potentially new host for the message's source component This produces a heuristic measure of affinity for a particular component to a particular processor Components are then migrated from over-loaded to under-loaded processors in a way that best satisfies the affinity measures for all components and processors but also considers the cost of component migrations The objective of the decision criteria is to balance load and achieve minimal network communications traffic Results are given but not compared to any optimal methods This problem more resembles a dynamic load sharing problem than an allocation problem

2.7. Performance Control of Distributed Telecommunication Service Platforms

We have identified the importance of scheduling methods for maximising the efficiency and performance of general distributed applications and have reviewed literature in the area, namely task allocation, load sharing and component allocation methods In this section, we consider the requirements of scheduling specifically in the context of telecommunication services executing on distributed platforms, with a view to applying scheduling methods for performance control In particular, we wish to find suitable solutions to the problem areas of optimal internal and external performance controls and optimal placement of Computation Objects in an IN/CORBA inter-working network, by considering suitable general distributed system methods We also require that these solutions take into account performance requirements specific to telecommunication services

2 7 1. Performance Requirements

Telecommunication services have a number of specific requirements in terms of performance From the user's perspective services are expected to display the following attributes

- Responsiveness Users generally expect telecommunication services to be highly responsive and long delays in accessing a service or slow response during service usage are not tolerable
- Stability and Reliability Each time a service is accessed, a similar performance is expected Consistency in the responsiveness of a service gives the user an impression of quality Only very small downtimes are tolerable as customer satisfaction is heavily influenced by even very infrequent service failures
- Fairness As all users generally pay the same amount for the same service, all users expect to be treated equally and experience the same responsiveness and reliability Also, users generally expect the performance of more costly service offerings to be better

From the service operator's perspective, the service network is expected to display the following attributes

- Responsiveness Delays need to be kept low to avoid aborted sessions and subsequent loss of revenue for the operator
- Stability and Reliability Apart from wishing to keep customers satisfied for good will and direct monetary reasons, large expenses may be incurred by the operator if regular maintenance is required due to unreliability of performance control mechanisms
- Optimality, Efficiency The performance control solution is expected to achieve the most efficient use of resources possible, maximising return on investment in service infrastructure
- Profitability Ideally a performance control mechanism should be able to relate resource usage and profitability of service types, whereby high profit (or high cost to the customer) services are assigned a greater proportion of resources, increasing the network capacity for processing high profit services or increasing service responsiveness
- Scalability and Flexibility of solution The performance control solution should not be closely tied to particular technologies and network topologies Upgrading of infrastructure should be possible without major reinvestment in performance control solutions Solutions would ideally be general and applicable across different platforms

Scheduling and load control solutions for telecommunication services should be mindful of these requirements in their design and applicability Considering these requirements, a number of technical aspects are implied for suitable performance control schemes We review these below particularly in relation to existing scheduling methods discussed previously

2.7 2 Possible Approaches to Performance Control for Distributed Telecommunication Services

Solutions to performance optimisation have generally focused on static deterministic scheduling schemes due to complexity issues The added complexity of component model interactions over the simpler job or task model has dissuaded researchers from considering the stochastics of service times and user traffic This is justified, as existing stochastic job allocation models are already complex in nature even with tight requirements on the product form nature of the model

A salient feature of work done on the stochastic job scheduling problem is that queuing delay variables must be separable, which is valid when jobs are independent (for example in [Tantawi and Towsley, 1985] and [Ross & Yao, 1991]) This aids formulation of a tractable optimisation problem Also, some reasonable conditions placed on the individual delays in the system ensure that the overall system response time is convex increasing These conditions allow solution with efficient numerical solution methods However, due to the added complexity of resolving queuing delay in networks with interdependent distributed components (and thus dependent queuing delays), even to an approximate degree, the problem of optimally assigning components based on an accurate estimation of queuing delay is not simple We are not aware of any attempts to do this and we discount stochastic schemes from investigation of optimal scheduling in component-based systems However, our hope for a responsive system is not all lost, as system delays may at least be influenced by balancing of load or maximising throughput in the network and, as we have seen from component allocation literature, these problems are linear and generally efficiently solvable

Several such component allocation schemes have been reviewed, however, none meets all our requirements Kihl [1998] does not account for the cost of remote communications, which can be a major performance factor in middleware systems Also, the model assumes that all component copies process the same amount of requests This assumption restricts possible solutions considered and, in general, will not give an optimal solution Anagnostou [1998] has a more comprehensive model, however, it is oriented towards minimising communications delays rather than limiting processor loading, which is our central concern He considers the cost of installation of multiple copies of components, which we consider to be a useful ideal Bastarnca *et al* [1998] do consider limits on loading but do not allow duplication of objects (which we consider a desirable feature) Their scheme is more akin to a classic task allocation problem Avramopoulos & Anagnostou [2002] also do not consider component copies in the network The scheme of Silaghi & Keleher [2001] is heuristic based and so non-optimal Most authors consider minimisation of some simple communications metric rather than maximisation of throughput and load limiting Processing costs associated with protocol

encoding/decoding have not been explicitly included in previous schemes None has considered biases for service profitability or user fairness

Dynamic scheduling has been considered in the literature mainly in the context of load sharing in job models This relates generally to what we have termed *internal performance control* However, due to the added complexity of dynamic schemes (readjusting the schedule in real-time) task allocation has generally not been considered in a dynamic context However, it is desirable that load control could somehow adapt to changing user demand in order to meet our requirements for stability, reliability and flexibility of the solution Little work has been done on this in relation to optimal dynamic scheduling in component architectures with the exception of component migration schemes, for example in [Avramopoulos & Anagnostou, 2002] However, optimal component migration schemes involve solution of NP-Hard problems at runtime and it is our view that more efficient methods are essential for practical solutions A lower complexity, optimal solution is required We consider the simple schemes of Kihl *et al* [1997, 1999] as too far from optimal for our purposes

Admission control (*external performance control*) has received relatively little attention in distributed systems literature but has been a focus of performance in telecommunication service networks (e g Intelligent Network load control) for a long time Admission control in a telecommunications environment is essential to ensure reliability and particularly stability during high load situations In the distributed systems domain, some recent work has considered application of admission control to web servers Chen & Mohapatra [2003] implement a *dynamic weighted fair sharing* (DWFS) scheduling algorithm specifically for controlling overloads in web servers Similar work has been done by Iyer *et al* [2000]

The issue of optimising profitability of a network by favouring high profit services has received some attention in the IN community but has escaped attention in the general distributed systems area In Intelligent Networks, several profit optimal schemes have been proposed, in relation to admission control schemes Jennings *et al* [1999] have considered a co-operative market and ant-based algorithms for optimising network profit based on service discrimination during over loads in INs Arvidsson *et al* [1997] have also considered profit optimal congestion control in INs based on an estimate of round trip delay It would seem natural to also apply such profit optimisation to component-based systems

2 7 3 Proposed Approaches

Having considered the general requirements and related work in the area, we consider the specifics of requirements for performance controls in distributed object-based telecommunication services

- ***Internal and External Performance Controls*** Admission control (external control) is traditionally viewed as an essential performance measure for telecommunication services and must be included in the performance control scheme An efficient internal control is also essential for proper functioning of the distributed platform As stated earlier, this external control should be integrated and co-ordinated in an optimal way with the internal control in order to meet the performance requirements

- ***Profit Oriented Optimisation*** Profit optimisation is now seen as a desirable property of a performance control in Intelligent Networks and should be included in performance schemes of future distributed object based networks However, profit optimisation should be balanced against fairness to users

- ***Dynamic Controls*** Changing demand from users or changing network conditions are expected to be handled in a controlled fashion in telecommunication services networks and this issue should be addressed Both internal and external performance controls should be dynamic, however, in order to be of practical use, they need to be computationally efficient and easily implemented

- ***Flexibility of Application*** Flexibility of the performance controls is important in respect to the following aspects Duplication of components (multiple component copies) in the system model should be allowed in order to gain the benefits of fault tolerance and load balancing There should be no artificial constraints on load balancing between duplicated components, for example by stipulating that all component copies receive the same load Multiple service classes with independent processing requirements should be considered Cost of component deployment, when there are multiple duplicated components, should be considered Aspects of component architectures, such as relationships between service specific components and common service independent components, should be considered The control should not be tied to a specific network topology or structure, that is, heterogeneity in node processing capacity and user traffic should be accommodated

- ***Accuracy of Control*** It is important that the main factors impacting on performance be included in the model to produce an accurate control For example, communication costs associated with protocol encoding and decoding times are significant in distributed systems and their impact should be included in models

- ***Load Centric Approach*** Telecommunication service performance generally considers optimisation of loading in networks, as opposed to considering communication delays This approach is dictated by the importance of avoiding overloads in environments with unpredictable and widely varying user service demands It is normally assumed (in Intelligent Network load controls for example) that the network is reliable (over-

capacitated and properly protected) and that the focus of attention should be efficiency and protection of processing nodes, rather than on the network infrastructure We apply this principle of separation of concerns and require that a load centric approach would also be desirable in component-based systems

These proposed approaches define a new set of performance problems for which we seek solutions in this thesis

2.7.4. Detailed Research Objectives

Our main research objectives in this thesis are as follows

- Development of suitable solutions to the proposed performance control approaches outlined above
- Verification that solutions to the proposed optimal approaches meet our original performance requirements for distributed telecommunication service platforms Critical examination and identification of deficiencies in this respect
- Comparison of our optimal approaches to our own best-effort non-optimal approaches and to existing methods to verify that a substantial gain is being achieved by the proposed optimal approach

In order to perform critical verification and comparisons the following related objectives are defined

- Development of a detailed component model for IN/CORBA inter-working that accurately expresses the architectures of the IN/CORBA gateway inter-working with a distributed telecommunication service architecture, namely the TINA service architecture
- Translation of this component model into suitable simulation and analytic models at an appropriate level of detail, in order to obtain quantitative verification and comparison of results These models should accurately express the salient performance issues in such architectures, namely processor loading and overloading, throughput, profitability, user fairness and service response times

2.8. Chapter Summary

We have given an overview of the evolution and future of telecommunication services and identified that technological and business drivers are moving service platform implementations towards the distributed computing model We have examined initiatives in this area, particularly in relation to inter-working with the Intelligent Network for medium term solutions and progression towards long term solutions The value of the TINA service architecture for structuring such solutions has been identified

Existing performance control methods for IN and TINA have been examined and the driving factors and requirements for telecommunication service performance controls identified Deficiencies in the area of performance control for distributed component-based telecommunication services have been identified Solutions to these deficiencies are sought in the general distributed systems performance literature and a number of useful methods identified Based on these methods we have proposed desirable telecom-centric properties of performance controls, which have not been seen in previous controls This essentially defines our problem area We conclude the chapter by stating our research objectives of finding suitable solutions and examining the solution behaviour in an IN/CORBA inter-working environment

Chapter 3. Methods and Tools

This thesis is primarily concerned with performance analysis and performance control of distributed telecommunication service networks This chapter introduces the mathematical and simulation methods and software tools used for the performance analysis conducted in Chapter 6 of this thesis Network performance is evaluated using discrete-event simulation techniques and supported by results from client-server analytic modelling These methods are described in §3 2 and §3 1 respectively Mathematical methods for the development of algorithms for network performance control are also outlined Control algorithms developed in Chapter 5 are based on mathematical optimisation techniques (Linear Programming and Mixed Integer Programming) and Market-based control techniques These methods are described in §3 3 and §3 4 respectively

3.1. Analytical Methods for Network Performance Analysis

Analytic network modelling is based on the application of *queuing theory*, a branch of mathematics which applies the theory of stochastic processes to the analysis of the behaviour of queuing systems This section provides a brief introduction to stochastic analysis and queuing theory with particular focus on methods suited to the analysis of client-server based systems Detailed introductions to stochastic analysis and queuing theory may be found in [Papoulis, 1984] and [Kleinrock, 1975] respectively

3.1.1 Basic Probability Theory

Probability theory concerns itself with describing *random events* The notion of *statistical regularity* is central to the theory This dictates that, under certain conditions, it is possible to make very precise statements about large collections of random events For example, if an unbiased coin is tossed many times, one expects that the outcome will be heads in approximately half of the cases In fact, the *probability* of a heads outcome, for an ideal unbiased coin, is exactly ½

More generally, consider an experiment having n possible outcomes, denoted o_1, ,o_n, where the outcome of the experiment cannot be predicted in advance An experiment of this kind is called a *random experiment* and the set of all its outcomes is called the *sample space*,

denoted $O = \{o_1, \ , o_n\}$ An event is the result of a single random experiment and comprises a subset A of the sample space A *probability measure* of event A, denoted $P[A]$, is a non-negative number indicating the likelihood of the occurrence of that event as the result of a single experiment, or alternatively, the expected frequency of occurrence of the event over multiple experiments Probabilities are defined so that the sum of the probabilities of all possible outcomes of an experiment sum to one, $P[O] = 1$

3 1 1 1 Random Variables and Distribution Functions

It is often the case that some value relating to an outcome is of more interest than the outcome itself This leads to the concept of a *random variable* The random variable X is a function, defined on the sample space O, which takes a value $X(o)$ for each $o \in O$ Random variables can be classed as *continuous* or *discrete*, depending on whether their *range* (the set of values they can take on) is discrete or continuous The probability that a random variable takes a certain value x is denoted $P[X = x]$ For discrete random variables this leads to the description of *a probability mass function* (pmf), denoted $p(x)$, as follows

$$p(x) = P[X = x]$$

Another convenient form for expressing the probabilities associated with a random variable is the *cumulative distribution function* (cdf) The cdf of a random variable X is defined as

$$F_X(x) = P[X \leq x]$$

and expresses the probability that X takes on a value less than or equal to x Where $F_X(x)$ has a continuous derivative everywhere, a related function, the *probability density function* (pdf), can be defined as follows

$$f_X(x) = \frac{dF_X(x)}{dx}$$

Note that

$$\int_{-\infty}^{\infty} f_X(x)dx = 1$$

Thus the pdf is a function which, when integrated over an interval, gives the probability that the random variable X takes on a value in that interval

3 1 1 2 Moments of a Random Variable

The probability distribution of a random variable is often characterised by a series of related parameters, called *moments* In most practical applications of probability theory, only the first

two moments are sought in order to approximate the characteristics of a random variable. Informally, the first moment gives the average value of the random variable and the second gives the spread of values around this average. In general, the k th moment of a random variable X, denoted $E[X^k]$, is defined by:

$$E[X^k] := \int_{-\infty}^{\infty} x^k f_X(x) dx$$

The first moment of a random variable X, denoted $E[X]$ or $\overline{X}$, and known as the *expectation*, or *mean*, or *average value* of X, is given by:

$$E[X] := \int_{-\infty}^{\infty} x f_X(x) dx$$

The second moment, the *variance* of a random variable X, denoted $V[X]$ or σ_X^2, is given by:

$$V[X] = \sigma_X^2 = \int_{-\infty}^{\infty} (x - \overline{X}) f_X(x) dx$$

3.1.1.3. Independent Random Variables

If we consider two random variables, X and Y, defined for some sample space, then the extension of the cdf for the two variables is defined as:

$$F_{XY}(x, y) := P[X \leq x, Y \leq y]$$

Associated with this function is a joint pdf, defined as:

$$f_{XY}(x, y) := \frac{d^2 F_{XY}(x, y)}{dx\, dy}$$

X and Y are said to be *independent* if and only if:

$$f_{XY}(x, y) = f_X(x) f_Y(y)$$

3.1.2. Stochastic Processes

A *stochastic process* (or *random process*) is a function $X(t,o)$, commonly denoted simply $X(t)$, of both time and probability space. For a fixed value of t it becomes a function of probability space, *i.e.* a random variable, whereas for a fixed value of o it is a function of time and is referred to as a *sample function* of the process. Stochastic processes are widely used to model the behaviour of telecommunication systems, for example, the number of callers on hold in a call centre queue can be modelled as a stochastic process (Figure 3.1).

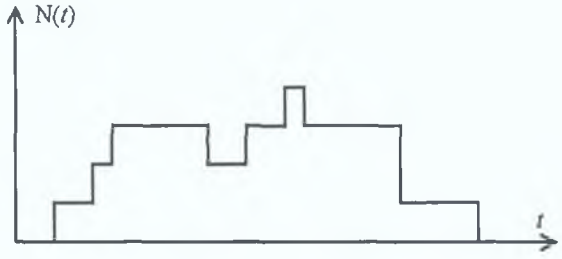


Figure 3.1: An Example Stochastic Process – The Number of Customers in a Queue over Time

The cdf of a stochastic process, denoted $F_X(x,t)$ is defined as follows:

$$F_X(x,t) := P[X(t) \leq x]$$

Furthermore, for n allowable values of t, $\{t_1,t_2,...,t_n\}$, a *joint cdf* may be defined for the process as follows:

$$F_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n;t_1,t_2,...,t_n) := P[X(t_1) \leq x_1, X(t_2) \leq x_2,..., X(t_n) \leq x_n]$$

The joint cdf is commonly denoted using the vector notation $F_X(\boldsymbol{x};\boldsymbol{t})$. In order to completely specify a stochastic process the values of $F_X(\boldsymbol{x};\boldsymbol{t})$ must be specified for all possible subsets of $\{x_i\}$, $\{t_i\}$ and all n. However, for many interesting and useful stochastic processes it is possible to provide this specification in very simple terms. In the following sections we list some classifications of stochastic processes based on their properties.

3.1.2.1. Stationary Processes

A stochastic process $X(t)$ is said to be *stationary* if $F_X(\boldsymbol{x};\boldsymbol{t})$ is invariant to shifts in time for all values of its arguments:

$$F_X(\boldsymbol{x};\boldsymbol{t}+\tau) = F_X(\boldsymbol{x};\boldsymbol{t})$$

where $\boldsymbol{t}+\tau$ is defined as the vector $(t_1+\tau,t_2+\tau,...,t_n+\tau)$.

All stochastic processes discussed in this thesis are considered to be stationary.

3.1.3. The Markov Process

The Markov process is the most important class of stochastic processes used in the analysis of telecommunication systems. It is a simple stochastic process in which the distribution of future states depends only on the present state and not on how it arrived in the present state. This simplification allows relatively easy analysis and yet the model is powerful enough to accurately model many aspects of performance in telecommunication systems. The most useful sub-classes of this process, which apply to this thesis, are described in the following sections.

Formally, a stochastic process is classified as a Markov process if and only if its next state is dependent only on its current state and not on any previous values This can be expressed mathematically as follows

$$P[X(t_{n+1}) = x_{n+1} | X(t_1) = x_1, X(t_2) = x_2, \ , X(t_n) = x_n] = P[X(t_{n+1}) = x_{n+1} | X(t_n) = x_n]$$

3 1 3 1 Markov Chains

A Markov process with a discrete state space is referred to as a *Markov chain* Markov chains can be either discrete-time or continuous-time For a discrete-time Markov chain the instants at which the state changes are preordained (a state transition takes place at each instant even if the state does not change as a result of the transition) For a continuous-time Markov chain the state transitions can take place at any instant in time Of particular interest is the random variable describing how long a Markov chain remains in its current state before a transition to another state occurs For a discrete-time Markov chain this time can be shown to be geometrically distributed, whilst for a continuous-time Markov chain it is exponentially distributed

3 1 3 2 Birth-Death Processes

A *birth-death process* is a (discrete- or continuous-time) Markov chain in which state transitions only take place between neighbouring states If, with no loss of generality, the set of integers is chosen as the discrete state space then the birth-death property requires that if $X_n = i$, then $X_{n+1} = i-1$, i or $i+1$ and no other value Birth-death processes play an important role in queuing theory, since they provide a means of modelling a queuing system where the time intervals approach zero (a continuous-time process), so that only a single event, an arrival or a departure, can occur during an interval Figure 3 2 illustrates the process

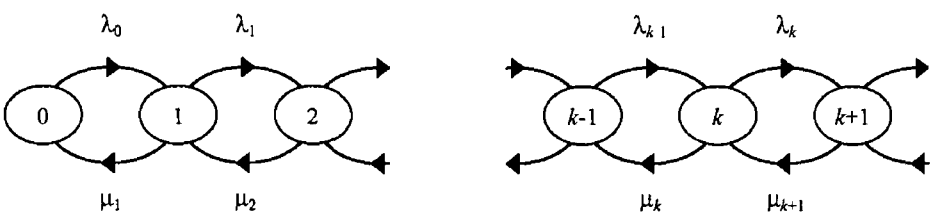


Figure 3 2 State Transition Rate Diagram for the Birth-Death Process

The probability of a birth-death process being in a particular state k at time t is denoted by $P_k(t)$, where

$$P_k(t+\Delta t) = P_k(t) - (\lambda_k + \mu_k)\Delta t P_k(t) + \lambda_{k-1}\Delta t P_{k-1}(t) + \mu_k \Delta t P_{k+1}(t) + o(t) \qquad k \geq 1$$

$$P_0(t+\Delta t) = P_0(t) - \lambda_0 \Delta t P_0(t) + \mu_1 \Delta t P_1(t) + o(t) \qquad k = 0$$

where λ_k is the *birth rate* (or *arrival rate*), representing the rate at which births (arrivals) occur when the population (number in the system) is k, and μ_k is the *death rate* (or *departure rate*), representing the rate at which deaths (departures) occur when the population (number in the system) is k The above equations can also be written in the form

$$\frac{dP_k(t)}{dt} = \lambda_{k-1}P_{k-1}(t) - (\lambda_k + \mu_k)P_k(t) + \mu_k P_{k+1}(t), \quad k \geq 1$$

$$\frac{dP_0(t)}{dt} = -\lambda_0 P_0(t) + \mu_1 P_1(t), \qquad k = 0$$

3 1 3 3 The Poisson Process

A special case of the birth-death equations above, in which the arrival rate λ_k is constant and the departure rate μ_k is zero in all states (i e $\lambda_k = \lambda, \mu_k = 0 \ \forall k$), yields the solution

$$P_k(t) = \frac{(\lambda t)^k}{k!}e^{-\lambda t} \qquad k \geq 0, \ t \geq 0$$

This is known as the *Poisson distribution* and describes a *Poisson process* The *Poisson process* is widely used in queuing theory for the modelling of arrival processes such as the sequence of times at which calls are originated by users of a telephony network

For a Poisson process, the average number of arrivals in $(0,t)$ is λt and the variance of the number of arrivals in the same time interval is also equal to λt The interarrival times of a Poisson arrival process are exponentially distributed, i e the pdf of the interarrival times is given by $f(t) = \lambda e^{-\lambda t}$, $t \geq 0$

The mean of the exponential interarrival time distribution is $1/\lambda$, while its variance is $1/\lambda^2$ The exponential distribution also exhibits the *memoryless* property, whereby the distribution of the time until a future arrival is independent of the time since the last arrival Therefore if, at some random time t, an estimate of the time that will elapse until the next arrival is evaluated then the result will be independent of the time that has elapsed since the last arrival

3.1 4 Bernoulli Trials

Bernoulli trials are a stochastic process widely used to model a sequence of independent generic trials that can result in two outcomes, success or failure, where the probability of success is p and the probability of failure is (1-p) Analytically we can describe the Bernoulli trials process with a sequence of indicator random variables $I_1, I_2, \ , I_n$, where the j^{th} indicator variable is used to describe the outcome of trial j Therefore we have

$$P[I_j = 1] = p, \ P[I_j = 0] = (1 - p)$$

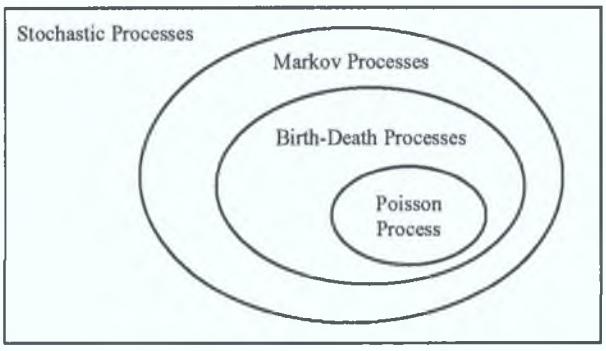


Figure 3.3: Relationship between Different Classes of Stochastic Processes

3.1.5. Queuing Theory

Queuing theory involves the study and analysis of the behaviour of queuing systems, where a queuing system is any system in which arrivals place demands upon a finite-capacity resource [Kleinrock, 1975]. Queuing theory is concerned with estimating values such as the mean queue size, mean waiting time or length of idle period, which are key metrics used for the evaluation of the performance of many systems.

In general, the length of a queue depends on the mean arrival rate (of *customers*), the mean rate at which arrival demands are serviced (the *service rate*) and on the statistical fluctuations of these rates. Clearly, when the mean arrival rate exceeds the system capacity the queue will grow in an unbounded manner. However, even where the mean arrival rate is less than system capacity, queues will sometimes grow due to clustered arrivals and/or variations in demands. The effect of these variations will be greater when the arrival rate approaches the maximum capacity of the system. We now introduce some of the basic terminology used in queuing theory.

In order to completely specify a queuing system the stochastic processes that describe the arrivals to the system and the structure and discipline of the servers must be described. The arrival process to a queue is typically described in terms of the probability distribution of the interarrival times of requests, denoted $A(t)$, where:

$$A(t) = P[\text{time between arrivals} \leq t]$$

The mean arrival rate to the queue is denoted λ, giving the mean interarrival time as $1/\lambda$. An arrival stream may be comprised of more than one *class* of arrivals, which may be described by different interarrival distributions.

The server process of a queue is typically described in terms of the probability distribution of the service times of request processed by the queue, denoted $B(x)$, where:

$$B(x) = P[\text{service time} \leq x]$$

The mean service rate of the queue is denoted μ, giving the mean service time $1/\mu$. It is possible for a queue to contain more than one server and it is possible that the distribution of service times will differ for each server.

A useful definition is that of *offered load* (denoted a) which is defined as the product of the arrival rate and the mean service time $a = \lambda/\mu$. The offered load is a dimensionless quantity that provides a measure of the demand placed on the system. It is normally expressed in units called *Erlangs*. A related quantity, the *load* of a queue, denoted ρ, is a measure of the proportion of time the queue server is busy, it is calculated as $\rho = \lambda/\mu$.

An important structural description of a queue is that of the *queuing discipline*, which describes the order in which requests are taken from the queue and allowed into service. Some common queuing disciplines are *First-In-First-Out (FIFO)* and *Last-In-First-Out (LIFO)*. Some queuing disciplines distinguish between classes of request arrivals on the basis of *priority*, with higher priority requests being granted preferential access to the server. The extent of storage capacity available in the queue to hold waiting requests may also be limited.

A fundamental result in queuing theory is *Little's Law*, which states that the mean number of requests in a queuing system (denoted $\overline{N}$) is equal to the mean arrival rate of requests to the system (λ), times the mean time spent by requests in the system (denoted T). $\overline{N} = \lambda T$

Queues can be classified according to the widely used shorthand notation $A/B/n$, where A describes the queue's interarrival time distribution, B describes its service time distribution and n is the number of servers in the queue. Values which A and B can take on include Markovian (i.e. exponential) (M), Deterministic (D), Erlangian (E) and General (G). Solutions to many combinations of interarrival time, service time distribution and queuing discipline are known for a single queue [Klemrock, 1975]. However, when queues are connected to form a network, analysis becomes more difficult.

3.1.6 Analysing Networks of Queues

In general, analytic models in which jobs departing form one queue arrive at another, or possibly the same queue, are called *queuing networks* [Gelenbe, 1999]. Unlike single queues, there is no simple notation for specifying the type of a queuing network. Certain subsets of the general queuing network have been identified and efficient exact analysis techniques developed. For more general networks, however, approximate techniques are the only practical solution. These topics are outlined below.

3.1.6.1 Product Form Networks

The simplest queuing network is a series of M single-server queues with exponential service time and Poisson arrivals. It has been shown by Jackson [1963] that each individual queue in

this series can be analysed independently of other queues The joint probability of the queue length of M queues can be computed simply by multiplying individual probabilties for each queue The queuing network is therefore termed a *product form network*

In general, the term applies to any queuing network in which the expression for the joint probability of queue lengths of M queues has the following form

$$P(n_1, n_2, \ , n_M) = \frac{1}{G(N)} \prod_{i=1}^{M} f_i(n_i)$$

where $f_i(n_i)$ is some function of the number of jobs at the ith facility and $G(N)$ is a *normalisation constant* (see [Gelenbe, 1999]) and a function of the total number of jobs in the system This property of product form networks renders them the simplest to analyse Important early work has shown that arbitrarily connected networks of queues, under certain assumptions, have product form solutions Jackson, Baskett, Chandy, Muntz and Palacios ([Baskett *et al*, 1975]) and Denning & Buzen [1978] have identified important classes of networks with product form solutions

At present, product form networks are the only class of queuing networks which have an exact solution in an explict form Furthermore, more general network models, such as the client-server model that has been investigated in this thesis, do not belong to this class However, product form networks do provide the basis for approximate algorithms which may be used to solve more general non-product form models These appoximate algorithms are described below

3 1 6 2 Mean Value Analysis (MVA)

Mean Value Analysis (MVA) is a simple solution technique that allows analysis of complex product-form queuing networks It is outlined here as it forms the basis for *Approximate MVA*, which is the core component of the analysis methods used in this thesis to solve complex non-product form networks

MVA gives only mean performance measures (mean delay and mean throughput) It can be applied to networks with a variety of service disciplines and service time distributions and can accommodate both single customer class and multiple class models Load dependent and load independent servers may also be represented

The MVA algorithm uses three key equations that are derived from *Little's Law* and the *Arrival Instant Theorem* Little's law states that the mean number of requests in a queuing system ($\overline{N}$) is equal to the mean arrival rate of requests to the system (λ), times the mean time spent by requests in the system (T) That is $\overline{N} = \lambda T$

The Arrival Instant Theorem states that, in a product-form queuing network, the queue length (A_{ck}) seen by a customer of class c on arrival at a service centre k is equal to the mean queue length Q_k there with the arriving customer removed form the network That is,

$$A_{ck}(\overline{N}) = Q_k(\overline{N - 1_c}) \qquad (3\ 1\ 1)$$

where $\overline{N} = (N_1, \ , N_c)$ is the workload intensity vector consisting of all class population sizes (N_c) and $\overline{N - 1_c}$ is the population $\overline{N}$ with one customer of class c removed The three equations from which the MVA algorithm is derived are

1 The service centre residence time for each chain

$$R_{ck}(\overline{N}) = D_{ck}\left(1 + A_{ck}(\overline{N})\right) \qquad (3\ 1\ 2)$$

where D_{ck} is the total demand of class c at centre k i e the product of mean service time and visit frequency

2 Applying Little's Law to the queuing network as a whole, the throughput (X_c) for each class is

$$X_c(\overline{N}) = \frac{N_c}{Z_c + \sum_{k=1}^{K} R_{ck}(\overline{N})} \qquad (3\ 1\ 3)$$

where Z_c is the think time for class c and N_c is the number of customers for class c

3 Applying Little's Law to each service centre, the mean queue length Q_{ck} for class c at centre k as well as the total mean queue length Q_k at centre k are

$$Q_{ck}(\overline{N}) = X_c(\overline{N}) R_{ck}(\overline{N})$$

$$Q_k(\overline{N}) = \sum_{c=1}^{C} Q_{ck}(\overline{N}) \qquad (3\ 1\ 4)$$

The MVA algorithm consists of finding an arrival-instant queue length A_{ck} and using this queue length to find the residence time (equation 3 1 2) The residence time is then used to derive the throughput (equation 3 1 3) Finally, from this throughput a new queue length may be found (equation 3 1 4)

There are two approaches to evaluating the equations, exact and approximate, which differ in the way the arrival instant queue lengths are computed In the exact method, applicable only to product form networks, equation 3 1 1 is evaluated exactly The trivial solution of the network for population 0 is used and applied to equations 3 1 2 and 3 1 4 From equation

3 1 4 the queue length for the next largest population is obtained The computation proceeds recursively over increasing populations until the target population is reached More detailed descriptions of exact MVA may be found in [Jain, 1991] and in [Lazowska *et al* , 1984]

3 1 6 3 Approximate Mean Value Analysis

As exact MVA requires an evaluation at every possible population, the computational complexity increases with the number of job classes and service centres An approximate method often becomes more practical for larger problems since it does not require evaluation of equations 3 1 2 and 3 1 4 for all populations Instead, the arrival instant queue lengths $A_{c\,k}$ are estimated based on the time averaged queue lengths at the service centres with the full customer population $\overline{N}$ and iteration is used to improve the estimate Many different functions may be used to estimate the arrival instant queue length The most commonly used is the Bard-Schweitzer approximation which assumes that $A_{c\,k}(\overline{N})$ is proportional to $Q_{c\,k}(\overline{N})$ The formulation for the approximation is

$$A_{c\,k}(\overline{N}) = Q_k(\overline{N - 1_c}) = \frac{N_c - 1}{N_c} Q_{c\,k}(\overline{N}) + \sum_{j=1\ j \neq c}^{C} Q_{j\,k}(\overline{N}) \qquad (3\ 1\ 5)$$

This method has been studied in [Wang & Sevcik, 2000] and its accuracy compared to that of exact MVA For networks with 3 job classes and 20 service centres, the error in queue length is reported at 1 45% while the error in response time is 1 04% Approximate MVA is a widely used analysis method and is a core component for solving sub-models in the Layered Queuing Networks model (discussed in next section), which has been used as the basis for analytical methods employed in this thesis

3 1 7 Layered Queuing Networks (LQNs)

Multi-tier client-server systems considered in this thesis pose particular problems for performance analysis methods described thus far The *Layered Queuing Network* model (LQN) [Woodside, 1996] extends the product-form model to reflect interactions between client and server processes The *blocking* nature of the remote procedure call in client-server systems causes problems for standard mean value performance analysis The remote procedure call is a type of *simultaneous resource possession* the requesting task and the serving task are both held by the same customer while the remote procedure call is in progress Furthermore, should the server continue to execute after the remote procedure call replies (a second phase of service), a second customer is effectively created These conditions preclude the direct application of the product-form model If the effect of simultaneous resource possession were ignored, the throughput estimates from a performance model would

be overestimated because the time needed to acquire resources would not be accounted for The LQN model allows these behaviours and permits layers of interacting clients and servers to be modelled

3 1 8 Solution Methods for LQNs

As mentioned above, multi-tier client-server application systems cannot be modelled directly using mean value analysis because the synchronisation blocking from nested sub-services is a form of simultaneous resource possession The problem can be solved using the *Method of Layers* (MOL) [Rolia & Sevcik, 1995] or the *Stochastic Rendezvous Network* (SRVN) [Woodside *et al* 1995] method The features of both have been combined in the *Layered Queuing Network Solver* (LQNS) [Franks & Woodside, 1998] These methods are described below

3 1 8 1 Method of Surrogate Delays

The *Method of Surrogate Delays* is a key concept in solving of replicated models of the type described below The method, [Jackson & Lazowska, 1982], is an approximate solution technique for queuing network models which have resources that are accessed simultaneously or have an overlap in possession Basically, the queuing network is split into multiple models In each model, a delay is obtained for a particular resource, modelled as a queuing station, while the other resources are represented by delay servers The method iterates the queuing delay estimates between the models until convergence

3 1 8 2 Stochastic Rendezvous Networks (SRVNs)

The *Stochastic Rendezvous Network* (SRVN), proposed by Woodside *et al* [1995], is used mainly to model a system with *software queuing* and *rendezvous*, although hardware elements may also be included in the model The model consists of an acyclic graph of clients and servers Clients and servers are collectively referred to as tasks, which are used to model users, devices, and software processes Requests from one task to another use the remote procedure call paradigm i e clients are blocked until the server responds The SRVN model is solved by first constructing a set of sub-models each consisting of only one server and a set of clients and their surrogate delays Next, the overall model is solved by applying MVA to each of the sub-models A variation of the Bard-Schweitzer MVA approximation is used where the waiting time expression is modified so that the queue length is found using arrival instant probabilities instead of simply scaling based on a fraction of customers in the system Throughput results from each sub-model are then used to adjust the surrogate delays in all of the other sub-models The solution iterates among all the sub-models until convergence criteria are met °

3.1.8.3 Method of Layers

The *Method of Layers* (MOL) [Rolia & Sevcik, 1995] solves client-server queuing networks by decomposing the network into a set of two-level MVA sub-models. Each sub-model forms a conventional product form queuing network where the servers form the stations and the clients form the customers. The MVA sub-model is constructed by splitting the input model into two sub-models, one for hardware contention and the other for software contention. The MOL algorithm then estimates the performance of the system under study by iterating among the various sub-models. It begins by solving the software sub-models from sub-model 1 to sub-model N-1. (There is no software sub-model N because the pure servers at level N make no requests). Once the software sub-models have converged, the performance results are used to set the think and service times for the tasks in the hardware model. The performance estimates from the solution of the hardware model are then used to set the service times for the various software sub-models. This sequence continues until the desired convergence criteria are met. Note that, unlike SRVN, the layering of servers is strict i.e. a server may only interact with servers in the next lowest layer.

3.1.9 The Layered Queuing Network Solver (LQNS)

The analysis methods employed in the Layered Queuing Network Solver (LQNS) [Franks & Woodside, 1998], combine the strengths of SRVN and MOL techniques to broaden the modelling scope and to improve the accuracy of solutions to Layered Queuing Networks (LQNs). The LQNS combines previous methods discussed i.e. the SRVN model and the Method of Layers. LQNs are solved using surrogate delays to solve the simultaneous resource possession problem arising from the nested calling pattern in the system being modelled. This goal is accomplished by partitioning the input queuing network into a set of smaller MVA sub-models, then iterating among these sub-models until convergence in waiting times. The solver software takes the LQN model specifications as input, in the form of *task*, *entry* and *processor* specifications, and returns a solution for the throughputs of tasks in the systems. We describe the model elements briefly below.

LQN models consist of layers of *tasks*, which are interconnected by their call patterns forming an acyclic task graph. Tasks represent interacting entities in the model that carry out operations and can also take on the properties of resources, including a queue, a discipline and a multiplicity. Tasks may represent hardware and software objects that may execute concurrently and are the central modelling entity in LQNs.

A task has one or more *entries*, which represent different operations that the task may perform. *Calls* are requests for service from a task entry to an entry of another task and *demands* are the total average amount of host processing and average number of calls required to complete a given entry. Calls may have asynchronous or synchronous behaviours.

Asynchronous calls do not wait for a reply from a called task whilst synchronous calls block the calling thread until it receives a reply

A task may have an associated host processor, which models the physical entity that carries out operations Tasks with a multiplicity greater than one can be used to represent multi-threading Tasks that do not receive any requests are called *reference tasks* and may be used to represent traffic sources or system users They cycle endlessly, creating requests to other tasks Tasks that do not have an associated processor, but merely model workload aspects of an object, are referred to as *pseudo tasks* More than one task may be associated with the same processor, in which case all such tasks share a common queue

Figure 3 4 shows a simple LQN model The larger parallelograms represent tasks and the smaller ones are the task's entries Circles represent physical resources, such as processors Directed arrows represent calls, with solid arrows of the type shown representing synchronous, or "rendezvous", calls In the task graph shown, *T0* is a reference task with a multiplicity of 10, which may be considered as representing 10 individual, but identically behaved, users connected to the system Task *T0* (a user) generates calls to entry *e1* of task *T1* at a given rate, and blocks until *e1* has executed Task *T1* is associated with processor *P1* and takes on the properties of the resource, namely a queue and queuing discipline Task *T1* has multiplicity 1 in this case (is single threaded) and thus represents a simple queue and service centre Note that, under certain assumptions, this LQN merely represents a closed product-form queuing network, which could be solved exactly by other means

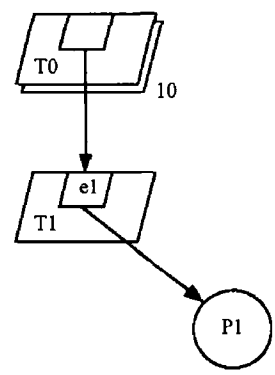


Figure 3 4 A Simple Layered Queuing Network

In Figure 3 5 below, a non-product form LQN model is shown In this case, a call from the user executes *e1* which subsequently executes *e2* whilst still blocking task *T1* In this case, processors *P1* and *P2* are being held by the same customer until *e2* has completed execution, that is, there is *simultaneous resource possession* in the system The effective service rates of the processors are not independent and thus the queuing network is not product-form The LQNS may be used in this case to obtain an approximate solution

Note that LQNs may model tasks that receive and generate calls to/from multiple other tasks Thus *fork-join* behaviour may be modelled and solved Systems with fork-join behaviour violate the fixed customer and routing assumptions in a closed queuing network and thus cannot be solved as a product-form network The model of interest to this thesis, given in Chapter 6, displays fork-jom behaviour and an LQN model has been constructed to obtain a solution

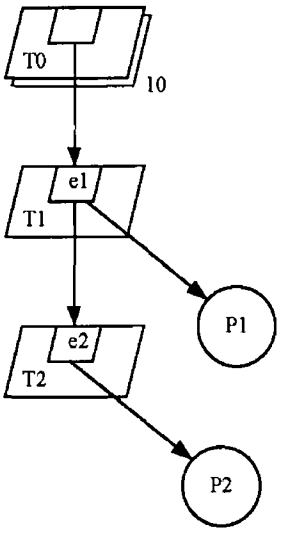


Figure 3 5 Simultaneous Resource Possession in an LQN

Note that LQNs can represent a finer level of detail within tasks, *activities* Activities are connected together to form a connected graph which represents one or more execution scenarios Execution may branch into parallel concurrent threads of control, which may or may not execute in parallel on the target system Execution may also choose randomly between different paths In Chapter 6, we use activities for modelling fork-joins and to represent choosing randomly between entries that a task may call This allows modelling of load sharing behaviour in the system

3.2. Simulation Methods for Network Performance Analysis

In general, simulation modelling is the process of designing a model of a real system and conducting experiments with this model for the purposes of

- understanding the behaviour of the real system
- aiding in the design or validation of the system
- determining strategies for effective operation and management of the system
- evaluating the performance of the system
- performing optimisation of the system

In the context of computer and telecommunication systems, a simulation is usually the execution of a model, in the form of a computer program, which gives information about the system being investigated Computer simulation is a technique that has gained widespread use in industry and is of fundamental importance in the design and evaluation of many types of systems

In this thesis, we are mainly concerned with simulation for evaluation of performance rather than with the system design process Specifically, simulation is used to evaluate and compare various algorithms for network performance control Performance investigations that are undertaken through simulation include, identification of system bottlenecks, analysis of steady-state behaviour of the system, analysis of the stability of the system and optimisation of system control parameters

Simulation is often used in conjunction with an analytic approach to performance evaluation, rather than as an alternative Analytic modelling often requires simplifying assumptions that render the results suspect until they have been corroborated by other techniques Simulation may also provide a wider range of performance metrics than is possible with mathematical analysis An almost arbitrary level of detail may be included in a simulation model whereas more complex analytic models may become intractable The down side to simulation is that it is often more time consuming to simulate than to analyse Also, simulation results are usually inexact, lying within some confidence interval rather than being exact values The following sections give a brief outline of common types of simulation model and identify a suitable method for simulation of computer and telecommunication systems

3211 Discrete vs Continuous Models

In continuous-valued simulation, the system state at any point in time is described by a set of continuous-valued state variables The evolution of the system state in time is usually characterised by a set of partial differential equations To implement a continuous-valued model as a simulation program, these differential equations are approximated by difference equations When time is incremented in the model, the simulation program computes new values for all system state variables Often iteration is required to converge to a solution for the new values Discrete-event simulation takes a fundamentally different view of how a system evolves The two most important differences are that **1)** the system state variables only take on discrete values and **2)** time may advance by fixed or variable amounts but state variables do not change within any interval over which time advances in a single step Discrete models are usually most useful for modelling of computer systems because the changes in system states (such as the arrival or departure of packets to and from a service station) occur at discrete points in time

3 2 1 2 Probabilistic vs Deterministic Models

If random variables are present, the model is *probabilistic* An appropriate density function must be specified for each random variable If variable values are always exactly known, then the model is *deterministic* Probabilistic models are normally the most useful for modelling computer systems as many of the underling processes (such as the arrival of customers to a system) are stochastic by nature and can be accurately modelled by standard probability distributions

3 2 1 3 Trace-driven vs Stochastic-driven Models

In trace-driven simulation, the model inputs are derived from a sequence of observations made on a real system The advantage of trace-driven simulation is that the model inputs are real world They are not approximations whose accuracy may be questionable Of course, such data is not always readily available In stochastic simulation, the system workload or the model input is characterised by various probability distributions During simulation execution, these distributions are used to produce random values, which are the inputs to the simulation model It is common in telecommunications modelling to assume that system inputs behave according to some standard stochastic process, such as a Poisson process

3 2 1 4 Stochastic Discrete-Event Simulation

Stochastic discrete-event simulation is normally the most suitable for modelling of telecommunication systems and is the method employed in this thesis As mentioned above, discrete-event simulation deals with system models in which changes happen at discrete instances in time, rather than continuously For example, in a model of a computer communications network, the arrival of a message at a router corresponds to a change in the state of the model The model state in the interval between successive message arrivals remains constant Since nothing of interest happens to the model between these changes, it is not necessary to observe the model's behaviour except at the time a change occurs

In a discrete-event model, *events* correspond to state changes and occur instantaneously The evolution of the simulation is described by a sequence of events and the times at which those events occur The change takes zero time i e each event is the boundary between two stable periods in the model's evolution (periods during which the state variables do not change), and no time elapses in making the change The model evolves as a sequence of events and to describe the evolution of the model, we need to know when the events occur and what happens to the model at each event

The heart of an event-driven simulation is the *event set* This is a set of (*event*, *time*) pairs, where *event* specifies a particular type of state change and *time* is the point in simulation time

at which the event occurs The event set is often implemented as a list, maintained in time-sorted order The first entry has an event time that is less than or equal to the event times of all other events in the list An event-driven simulation also maintains a simulated time clock, the value of which is the (simulated) time of the most recent event that has occurred The basic operation of an event-driven simulation, with the event set implemented as a sorted list, is as follows

1 Set the simulation clock to 0 Place a set of one or more initial events in the event list, in time-sorted order

2 Fetch the event E consisting of the ordered pair (E type, E time) at the head of the event list If the event list is empty, terminate the simulation

3 Set the simulation time to E time If E time is greater than the maximum simulation time specified for the execution of the simulation model, terminate the simulation

4 Use the event identifier E type to select the appropriate event-processing code

5 Execute the selected code During this execution, an event may update system information held in global data structures, and it may cause a new events E' (with E' time > E time) to be inserted in the event list Note that it does not change simulation time

6 At the completion of execution of the event code, go to 2

A key point is that events never change the simulation time directly They can only affect simulation time by the creation of new events that are inserted into the event list A more detailed description of discrete-event simulation may be found in [Banks, 1998]

3 2 1 5 The OPNET Simulator

OPNET Modeller™ is a hierarchical, object-oriented development environment that is designed specifically for the modelling and analysis of communication networks It is based on the principles of stochastic discrete-event simulation described above It provides a hierarchical graphical interface for model specification in which *network*, *node*, *process* and *link* models are combined to realise a complete system model It also provides a range of tools for the specification of simulation inputs and filtering and analysis of outputs

OPNET network models define the position and interconnection of communicating entities, or *nodes* Each node is described by a block structured data flow diagram, or OPNET node model, which typically depicts the interrelation of processes, protocols and subsystems Each programmable block in a node model has its functionality defined by a *process* model, which is defined by means of C programming code encapsulated within a graphical state-transition diagram Specification of processes in C is facilitated by an extensive library of support functions providing a range of simulation services All simulation models described in this

thesis were implemented using the OPNET simulation environment Further details of the simulator are available on the Web [OPNET, 2004]

An example of a process model (the core structure of OPNET model behaviour) is shown in Figure 3 6 below This example shows the main elements of a simulation model of a simple queuing system that has been realised in the OPNET simulation environment

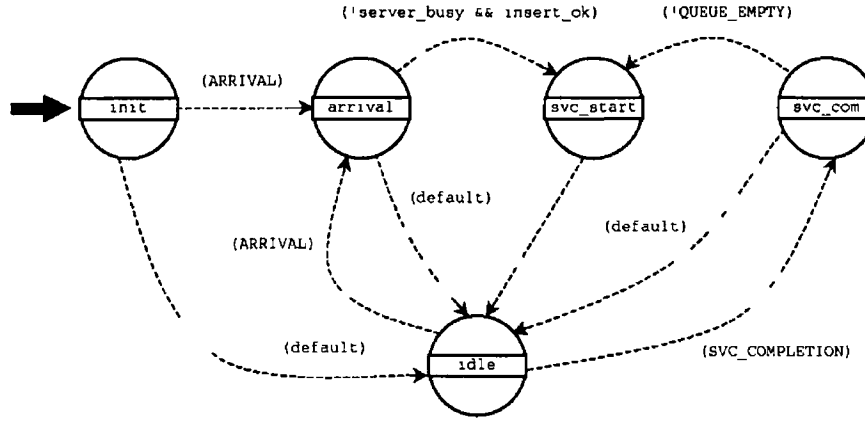


Figure 3 6 An Example OPNET Process

The model shown represents a simple *First In First Out* queue The circles represent system *states* and arrows represent *transitions* between states The system may only be in one state at any given instant and may only move to a new state via a transition Each transition has an associated condition that must be fulfilled in order to change state Transitions may be dependent on system *events* (such as arrival of a packet or completion of service of a packet) Each state may execute instructions when entered and exited This code may effect a change in *state variables* in order to affect which transition is taken out of the current state An example of a state variable in our example is QUEUE_EMPTY

The idle state is often central to the model Generally, a system will remain in an idle state until an event occurs In this case, the events are (ARRIVAL) which indicates and arrival of a packet and (SVC_COMPLETION) which indicates that a packet has completed service The arrival event is generated by an external process not shown here When an arrival occurs, the model makes a transition to the arrival state and processes the arriving packet If the server is idle, then the packet is processed immediately in the (svc_start) state, otherwise the packet is queued to be processed later and the system returns to the idle state Processing of the packet simply involves setting an event some time in the future for the end of the service time Once this is done the system returns to the idle state and waits for the end of service event, (SVC_COMPLETION)

3.3. Mathematical Programming Methods for Network Performance Control

This section describes the mathematical programming methods that are used to formulate and solve the optimal object placement and network control problems of Chapter 5 Specifically, these problems fall into the categories of *Linear Programming* and *Mixed Integer Programming* A general survey of the methods is given in this section As the methods are to be applied to real-time control problems, consideration is also given to the practicality and the complexity of solution computation Suitable software tools for problem solution are also discussed

3 3.1. Mathematical Programming

In a *mathematical programming* or *optimisation* problem, we seek to minimise or maximise a real-valued function of real or integer variables, subject to a certain set of *constraints* on the variables The function to be optimised is referred to as the *objective function* The possible values of the objective function, subject to the set of constraints, form a *feasible region* An optimisation problem solution gives values of the problem variables, which produce a maximum or minimum value of the objective function in the feasible region, if such a value exists Optimisation problems are generally classified as *linear* or *non-linear* A problem is classified as linear if the objective function and all constraints are linear All problems considered in this thesis are linear Linear problems that have all real variables are referred to as *Linear Programming* (LP) problems while linear problems with both real and integer variables are referred to as *Mixed Integer Programming* (MIP) problems Details of LP and MIP problems and their solution methods may be found in [Walsh, 1985] and [Schrijver, 1986]

3.3.2 Linear Programming

The *general linear programming problem* may be expressed in vector-matrix notation as

minimise or maximise

$$z = \mathbf{c}_0\mathbf{x}_0, \qquad (3\ 3\ 1)$$

subject to the constraints

$$\mathbf{A}_0\mathbf{x}_0 \leq, =, \geq \mathbf{b} \qquad (3\ 3\ 2)$$

and to the non-negativity restrictions

$$\mathbf{x}_0 \geq \mathbf{0}, \qquad (3\ 3\ 3)$$

where $\mathbf{c}_0^t$ is a *n*-component row vector of real constants, $\mathbf{x}_0$ is a *n*-component column vector of real variables (the *problem* or *decision variables*), $\mathbf{A}_0$ is a *m* x *n* matrix of real coefficients and **b** is a *m*-component column vector of real constants. z is referred to as the *objective function*. Note that vector inequalities are applied on a component by component basis. In principle, all problems of this form can be solved in finite time, provided that a solution exists. It is possible that problems of this form are *infeasible* (do not have a solution in the feasible region) or are *unbounded* (the value of the objective function may increase or decrease arbitrarily within the feasible region).

3.3.3. Linear Programming Solution Methods

Three principal mathematical methods exist for the solution of LP problems. The *simplex method*, devised by Dantzig [1953], is the basis of most LP solution methods available today. Although the theoretical efficiency is poor (an exponential-time algorithm), in practice the method performs efficiently for most practical problems and may be efficiently implemented on a computer system. Indeed research has shown, as discussed in [Lagarias & Todd, 1990], that a probabilistic analysis reveals the practical efficiency of the simplex method to be polynomial.

The *ellipsoid method*, devised by Khachiyan (see [Aspvall *et al.*, 1980]), is a polynomial-time algorithm but is generally considered impractical to implement as the operations performed by the algorithm would require a precision higher than that normally available on a computer system. Also, in contrast to the simplex method, the number of iterations required to solve a problem is very close to the theoretical upper bound, so that in practice it may not perform significantly better than the simplex method.

The more recent *interior-point* methods, [Karmarkar, 1984], provide an efficient alternative to the simplex method and are generally preferred for implementing very large-scale problems.

3.3.3.1. The Standard Simplex Method

The simplex method is an iterative procedure for solving the general linear programming problem. A geometric interpretation of the simplex method is that, given that the feasible region represents a polyhedron, the algorithm moves from vertex to vertex along edges until an optimal vertex is reached. The main elements of the method are described below. Further detail may be found in [Walsh, 1985].

The first step of the simplex method is to change all inequality constraints in (3.3.2) into equality constraints. A *slack variable* is added to the left-hand side of $\leq$ inequality constraints and, similarly, a *surplus variable* is subtracted from the left-hand side of $\geq$ inequality constraints. Assuming that the original problem is rearranged so that the first *a*

constraints are $\leq$, the next b constraints are $\geq$ and the remaining $(m - a - b)$ are equality constraints, the constraint equations of (3 3 2) may be written in the form

$$\mathbf{A}_0\mathbf{x}_0 + \begin{pmatrix} \mathbf{I}_a & \mathbf{0} \\ \mathbf{0} & -\mathbf{I}_b \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \mathbf{x}_s = \mathbf{b}, \qquad (3\ 3\ 4)$$

where $\mathbf{x}_s = [x_{n+1}, \ , x_{n+a+b}]$ is the vector of surplus and slack variables, $\mathbf{I}_a$ and $\mathbf{I}_b$ are unit matrices of orders a and b The non-negativity restrictions of (3 3 3) are now

$$\mathbf{x}_0 \geq \mathbf{0}, \ \ \mathbf{x}_s \geq \mathbf{0} \qquad (3\ 3\ 5)$$

and the objective function of (3 3 1) becomes

$$z = \mathbf{c}_0^{'}\mathbf{x}_0 + \mathbf{c}_s\mathbf{x}_s \qquad (3\ 3\ 6)$$

where $\mathbf{c}_s$ is the zero vector with $(a + b)$ components The problem of maximising the objective function of (3 3 6) subject to (3 3 4) and (3 3 5), is equivalent to the original problem defined by (3 3 1), (3 3 2) and (3 3 3) The original problem may therefore be expressed in the form

Maximise $\quad z = \mathbf{c}\,\mathbf{x},$

subject to $\quad \mathbf{A}\,\mathbf{x} = \mathbf{b} \ \ \textit{and} \ \ \mathbf{x} \geq \mathbf{0}, \qquad (3\ 3\ 7)$

where $\quad \mathbf{A} = \begin{pmatrix} & \mathbf{I}_a & \mathbf{0} \\ \mathbf{A}_0 & \mathbf{0} & -\mathbf{I}_b \\ & \mathbf{0} & \mathbf{0} \end{pmatrix}, \ \ \mathbf{x} = \begin{pmatrix} \mathbf{x}_0 \\ \\ \mathbf{x}_s \end{pmatrix}, \ \ \mathbf{c} = \begin{pmatrix} \mathbf{c}_0 \\ \\ \mathbf{0} \end{pmatrix}$

This form of the problem gives the starting point for the simplex algorithm Note that if m, the number of constraints, is equal to N, the number of variables, and if rank(**A**) = m, then equations (3 3 7) have the unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ and there is no optimisation problem The feasible region, if it exists, consists of a single point Also, if $m > N$ and rank(**A**) = N, then $(m - N)$ of the constraint equations are redundant and again equations (3 3 7) have a unique solution However, assuming that $m < N$ and rank(**A**) = m, then the problem forms an optimisation problem where equations (3 3 7) may have a non-unique solution (the feasible region is some region greater than a single point)

Algorithm Iteration

Assuming an optimisation problem as described above, if m linearly independent column vectors of **A** are chosen, and if $(N - m)$ variables corresponding to the remaining columns of **A** are set equal to zero, then the resulting set of m equations has a unique solution, termed the

basic solution The m variables of the basic solution are termed *basic variables* while the remaining $(N - m)$ variables are termed *non-basic variables* The column vectors of **A** corresponding to the basic variables together comprise the *basis* matrix It has been proven that in searching for an *optimal feasible solution* of the general problem, it is only necessary to consider *basic feasible solutions* The simplex algorithm progresses by moving from one basic feasible solution to another that gives an improvement in (or the same value of) the objective function Eventually the iterations lead to an optimal basic feasible solution, if one exists Note that it is also possible that the problem is *unbounded*, whereby there is no upper bound on the objective function

In order to progress from a basic feasible solution to a better one the following procedure is applied **1)** Determine which non-basic variable will increase the objective function value most swiftly if allowed to take on a positive value This variable is moved to the set of basic variables **2)** Allow this new basic variable to increase in value until one of the basic variables is forced to zero This variable is moved to the set of non-basic variables The solution of the new basis, formed by the above steps, gives the new basic feasible solution for the next iteration Iteration is stopped when there is no non-basic variable that, if allowed to become positive, would increase the value of the objective function (assuming a feasible solution exists)

Algorithm Initialisation

To commence iteration of the algorithm, an initial basic feasible solution is required If the *basis* of the initial problem of (3 3 7) has a feasible solution, simplex iteration commences from this solution These problems are referred to as *single-phase* problems If there is no feasible solution then an additional phase is required to find an initial basic feasible solution An auxiliary problem is formulated whereby *artificial variables* are added to the constraint equations In the *two-phase* method, the artificial variables are given a price coefficient of 1 for a minimisation problem or –1 for a maximisation problem and all other variables are given a price of zero The objective function becomes the sum (or negative sum) of the artificial variables An initial feasible solution to the auxiliary problem is readily available and the simplex calculations are applied to produce a basic feasible solution This completes *Phase 1* of the method In *Phase 2*, the non-artificial variables are then reassigned their original cost coefficients (c vector) and simplex calculations proceed normally Several variations of this method have been devised for example the *M*-method, [Charnes, 1953]

3 3 3 2 The Dual Simplex Method

Normally an initial basic feasible solution is required to initiate the simplex method This can consume a considerable amount of computation if the introduction of artificial variables is

necessary The *dual simplex method* has the advantage of allowing initiation with a non-feasible basic solution The fundamental idea behind the method is that the choice of basic and non-basic variables to be exchanged is determined by criteria applied to the current *dual tableau* (see [Walsh, 1985]) Once these variables have been chosen, the usual simplex transformation equations are used Generally, duality is useful in the following situations **1)** There are more constraints than variables The dual basis matrix is then smaller than the primal basis matrix and so computation is reduced, **2)** The dual constraints are all of the <= type A basic feasible solution for the dual problem can then be written down immediately **3)** It is required to add a further constraint to a problem already solved The additional primal constraint becomes merely a further variable in the dual problem, with a value of zero at the time it is added Further details of the dual simplex method and its applications are available in [Walsh, 1985]

3 3 3 3 Efficiency of Simplex Methods

As mentioned previously, the simplex method is an exponential-time algorithm i e in the worst case the number of possible arithmetic steps required to reach a solution increases exponentially with the number of problem variables However, it has been shown that the probable average running time of the simplex method is much better (polynomial-time bounded) In practice, the simplex method is generally considered efficient when applied to large practical problems and indeed most current software implementations for the solution of industrial-scale linear programming problems are still based on the simplex method The use of the dual simplex method is often applied judiciously by software implementations to reduce computation in two-phase problems

3 3 3 4 Interior-Point Methods

Recently *interior-point methods*, a new class of polynomial-time methods for the solution of linear programming problems, have been the subject of much research Since its original inception, [Karmarkar, 1984], refinements of the method have been studied and implemented on computer systems and are reportedly competitive with the best simplex methods available For very large-scale problems, interior-point methods may even outperform simplex methods The general idea behind interior-point methods is that the algorithm generates iterates that lie in the interior of the feasible region (rather than strictly on the boundary as simplex methods do) The iteration then progresses toward the boundary of the region and towards an optimal solution Although interior-point methods promise improved efficiency for very large-scale LP problems, the problems in this thesis are considered small enough in scale to be handled efficiently by simplex methods

3 3.4 Mixed Integer Programming

When a programming problem has decision variables, which may only take integer values, it is referred to as an *Integer Programming* problem When some, but not all of the decision variables are restricted to integers, the problem is referred to as a *Mixed Integer Programming* (MIP) problem The usual LP solution methods cannot be applied directly to such problems and often far more computation is required than for the same problem without integer constraints The sections below describe relatively efficient solution methods for such problems (Note that all MIP problems considered in this thesis are linear)

3 3 5. Mixed Integer Solution Methods

The most common approaches to the optimal solution of MIP problems are the *branch and bound* and the *branch and cut* methods These methods rely on LP solution methods (such as the simplex method) to solve sub-problems, which have had the integer constraints relaxed Further details of these methods may be found in [Schrijver, 1986]

3 3 5 1 Branch and Bound

The most widely used method for solving integer and mixed integer programs is *branch and bound* This method begins by finding the optimal solution in the absence of the integer constraints If it happens that in this solution the decision variables whose values are constrained to be integers already have integer values, then no further work is required If one or more integer variables have non-integral solutions, the branch and bound method chooses one such variable and "branches", creating two new sub-problems where the value of that variable is more tightly constrained These sub-problems are solved and the process is repeated, "branching" as needed on each of the integer decision variables, until a solution is found where all of the integer variables have integer values (to within a given tolerance) Hence, the branch and bound method may solve many sub-problems, each one of which is an LP problem The "bounding" part of the branch and bound method is designed to eliminate sets of sub-problems that do not need to be explored because the resulting solutions cannot be better than the solutions already obtained

3 3 5 2 Branch and Cut

With the *branch and cut* method, a lower bound is provided by the LP relaxation of the integer program If the optimal solution to the LP problem is not integral, this algorithm searches for a constraint, which is violated by this solution, but is not violated by any optimal integer solutions This constraint is called a *cutting plane* When this constraint is added to the LP the new optimal will be different, potentially providing a better lower bound Cutting planes are evaluated iteratively until either an integral solution is found or it becomes

impossible or too expensive to find another cutting plane In the latter case, a branch operation is performed and the search for cutting planes continues on the sub-problems

3 3 5 3 Efficiency of MIP Solution Methods

Unlike LP problems, MIP problems generally exhibit an extremely large (combinatorial) increase in the number of possible solutions as the problem size increases However, the branch and bound method needs only to enumerate a fraction of the feasible solutions to reach an optimal integer solution Also, if integer variables are restricted to *binary* (0-1) variables, computation may be further reduced All MIP problems in this thesis are formulated with only binary variables The branch and bound method is generally regarded as the most efficient method for these problems and is the one adopted here Branch and bound solving software, such as that described below, often includes proprietary refinements, which further reduce solution complexity

3 3 6 Mathematical Programming Solvers

A great number of implementations of programming solvers exist at present The IBM *Optimisation Solutions and Library* (OSL) was chosen from amongst these because

- it provided efficient implementations for all problem types that were encountered
- it is easily integrated with other software (e g simulations) via a C language API
- it is well recognised in industry, is well documented and is relatively easy to use

The OSL comprises of an optimal set of functions for easily creating, manipulating, solving and analysing linear, mixed-integer and quadratic programming models The LP problem solver includes simplex method and interior-point solvers The simplex method solver was chosen, as the problems encountered were relatively small-scale The MIP problem solver uses the branch and bound method of solution Further details of the OSL may be found on the Web [OSL, 2004]

3.4. Market-Based Methods for Network Performance Control

Market-based control is a distributed resource allocation and control technique, which aims to achieve some overall coherent global behaviour of a system, through the use of economic models The resources (and the use of resources) in the system are modelled by *supplier* and *consumer agents*, which have individual goals A consumer attempts to optimise its performance criteria by obtaining the resources it requires, without concern for system-wide performance A supplier's goal is to optimise its individual profit, based on its choice of resource allocations to consumers, again without concern for system-wide performance

Economic models often introduce money and pricing as the technique for co-ordinating the selfish behaviour of the agents The price a supplier charges for a resource is determined by its supply and by the demand from the consumers for the resource Typically, each agent participates in an iterative auction process where it faces a set of prices and replies with a demand/supply message From the total demand/supply of the market, a new set of prices is computed This is iterated until supply is equal to demand for each commodity, referred to as the *general equilibrium* of the market (illustrated in Figure 3 7 below)

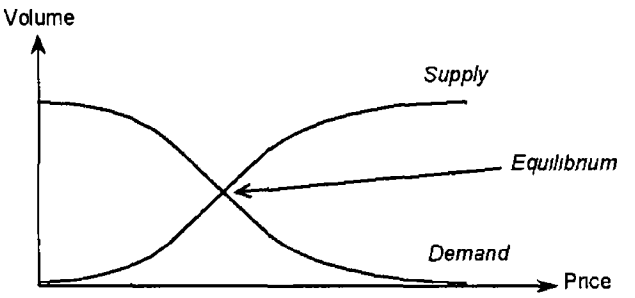


Figure 3 7 General Market Equilibrium

An alternative to the *price-oriented* approach is the *resource-oriented* approach In this case, the general equilibrium is expressed as the allocation of resources such that each agent is willing to pay the same price for an additional small amount of resource The auctioneer sets the allocation of commodities to agents at each iteration and agents report how much they are prepared to pay for an additional small amount of each commodity The auctioneer then takes these declarations into account when changing the allocation in the next iteration - agents with high willingness to pay get more, the others less The algorithm terminates when, for all commodities, all agents are willing to pay the same price for an additional small amount of the commodity

In order to model the bidding behaviour of agents, *utility functions*, which encapsulate the preferences of a consumer, are employed A utility function is essentially a preference ordering, where a high value of the utility function for some consumption bundle means that such a bundle is preferred over a bundle with a lower utility function value If a number of agents can trade commodities in such a manner that all agents have higher utility after the trade, then the agents are motivated to trade commodities with each other If trading is performed in the context of a price-based market (every commodity is evaluated in terms of another commodity or using a monetary unit) each agent will face the optimisation problem of how to maximise its utility given the prevailing market prices and utility function For the resource-oriented approach, only the utility functions are required to perform the auction process

Market-based approaches to resource allocation hold the advantages of the decentralised control approach They can facilitate resource allocation with very little information e g price A coherent global behaviour may be achieved through very simple interactions e g trading and auctioning Agents require only very limited knowledge of each other and are thus more dynamic than a centralised controller However, the market-based approach doesn't generally guarantee an optimal solution but can achieve adequate results A review of market-based control for resource allocation may be found in [Clearwater, 1996]

In Chapter 5, a resource-oriented market strategy is developed and applied to resource allocation problems on distributed service platforms Utility functions are derived from the revenue value generated for successful service sessions and resources are allocated in order that overall network profit is maximised

3.5. Chapter Summary

The chapter has detailed the methods and tools required to perform the analysis and simulation work undertaken in this thesis In particular, a suitable analysis model (LQNs) has been chosen for application to the analysis of client-server systems Also a suitable simulation method (discrete-event simulation) and simulation tool (OPNET) have been identified

Mathematical programming methods, that are applied to the network optimisation and control problems of Chapter 5, have also been discussed An overview of the Market-based methods applied in Chapter 5 have been given Various other concepts which arise in the thesis (basic probability theory, stochastic processes and queuing theory) have been outlined

Chapter 4. Model of a Distributed Telecommunications Service Platform

In this chapter we develop a simulation model of a distributed CORBA-based service platform executing three different IN services This model is employed to study the performance controls developed in Chapter 5 The service platform model is based on an IN/CORBA Computational Object model, which we describe here We give our modelling assumptions and describe the operation of the simulation model in detail This service platform model is also used to derive an analytic model in Chapter 6

4.1. Simulation Model Description and Rational

The primary motivation for the simulation model is to investigate how loading and delay associated with a CORBA-based service platform varies as a function of incoming traffic intensity, Computation Object placement, and external and internal performance controls

The simulation model structure is based on TINA-IN and IN/CORBA inter-working, as are described in Chapter 2 The basis of the model is the replacement of the Intelligent Network *Service Control Point* (SCP) and *Service Data Point* (SDP) with a network of service nodes which host software objects communicating via a remote method call mechanism, i e via the CORBA Object Request Broker (ORB) In this scenario, the IN *Service Control Function* (SCF) and *Service Data Function* (SDF) are no longer encapsulated within single functional entities, but are decomposed into fine-grained *Computational Objects* (COs) which use the ORB for communication These objects communicate with entities in the legacy Intelligent Network via *IN/CORBA Gateways* Thus, the service logic programs and data that normally reside at the SCP and SDP are distributed across a multi-node network It is the performance of this distributed SCP/SDP that is the primary target of investigation Figure 4 1 shows the general network configuration in the CORBA-based SCP/SDP scenario and how it may interconnect to a legacy Intelligent Network

In order to determine performance characteristics of such a network with accuracy, a certain level of detail is required in the model The performance effects of distributed communication between objects in the CORBA domain, which provide IN services, are of primary importance Thus, individual procedure calls between distributed objects are modelled in detail As loading is heavily dependent on the nature of the service being executed in the network, attention has been paid to accurately representing a number of real IN services in the model These considerations allow an accurate representation of loading and how it is distributed in the network In order to accurately determine delays in the network and the overall delays experienced by users, the effects of queuing in processing nodes needs to be modelled The semantics of message processing by the ORB have been taken into account in determining the server model The remainder of this chapter details these model elements and states the assumptions made in arriving at their particular representations A description of how the model was implemented in OPNET is also given

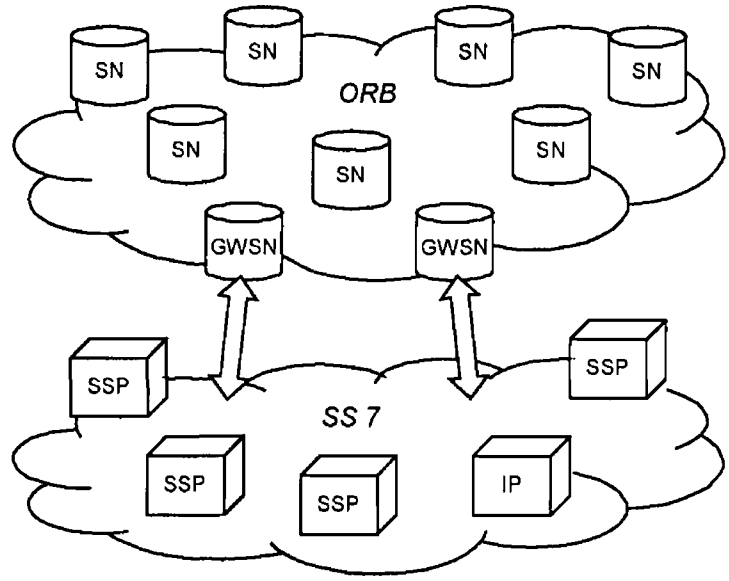


Figure 4 1 IN/CORBA Interworking Scenario

4.2. The Network Model

The network configuration chosen for study consists of a network of ten CORBA processing nodes Two of these ten nodes have an interface to the SS 7 domain These two nodes are referred to here as *Gateway Service Nodes* (GWSNs) The remaining eight CORBA *Service Nodes* (SNs) do not interface directly with the SS 7 domain The motivation for the number of SNs and GWSNs chosen is based on the following considerations

- It is assumed that the CORBA service network will replace one collocated fault-tolerance pair of SCP nodes that interact with one SDP in a legacy Intelligent Network

- Apart from the gateway function of the GWSNs, GWSNs and SNs are considered identical and all ten service nodes are assumed to have equal processing capacities
- It is assumed that individual CORBA service nodes have somewhat less processing capacity than a purpose-built SCP/SDP as they would likely consist of less costly, generic Pentium based architectures It is assumed that the processing capacity of a single service node is 0 4 times the processing capacity provided by one SCP interacting with the SDP
- It is also assumed that service execution in the CORBA domain requires considerably more processing time than in an SCP/SDP due to the additional processing required for distributed calls and the added complexity of the service architecture We assume that each CORBA node has only 50% the efficiency of one SCP/SDP

Given these assumptions, 10 service nodes are required to provide approximately the same processing capacity of the original two legacy SCPs and one SDP Numerically
$(0\,4\times0\,5\times10\ SNs\equiv2\ SCP/SDPs)$

Furthermore, given that this network of service nodes replaces two SCPs, it was considered reasonable that each legacy SCP would be replaced by one GWSN

The following additional assumptions were made regarding the network

- The service nodes are assumed to be fully connected by a highly redundant network with low transmission times
- It is assumed that delays in network transmitter queues and transmission times in the network are negligible compared to delays due to marshalling and de-marshalling of CORBA method calls between nodes Experiments with an IN/CORBA prototype have shown [McArdle *et al*, 2000] that marshalling and de-marshalling times for the IN/CORBA IDL used for this model are typically an order of magnitude greater than transmission times over a fast LAN As service session IDL is similarly complex, we assume that this is also the case for the service COs
- It is assumed that the transport is reliable, i e there is no message loss
- The legacy IN entities (the SCP and IP) and the SS 7 network are not modelled explicitly but are viewed as an amalgamated source and sink of messages arriving to and departing from the GWSN nodes Messages sent to the SSP are simply delayed before a return to the gateway is made
- As the service network performs functions normally provided by the IN SDP, it is assumed that no legacy SDPs are required in the SS 7 domain and interactions with legacy IN SDPs are not modelled
- It is assumed that all Intelligent Peripheral (IP) functions remain in the SS 7 domain as these are normally tightly coupled to the switching network (SSPs) Communication with these is modelled (for services that require the IP)

4.3. Modelling the Gateway and Service Components

In order to define message sequences for service execution in the model, we start by defining the objects required at the gateway and in the CORBA domain and consider their interactions during execution of a service session The model of the gateway and service components defined here has been derived from the TINA-IN and IN/CORBA inter-working studies detailed in Chapter 2

4.3 1 Gateway Components

The GWNS nodes execute the functionality required for inter-working between SSPs and IPs in the SS 7 domain and the CORBA-based SCP/SDP It is assumed that the Gateway function consists of the standard IN/CORBA inter-working components [OMG, 1999], described in §2 4 2 The IN/CORBA Gateway function is modelled by considering only the core inter-working components necessary for communication between the IN and CORBA domains during a service session i e the SSF Proxy, the IP Proxy and the SCF Proxy objects (Figure 4 2) The factory objects and the semantics of object creation are not modelled as it is assumed that this process can be handled efficiently with little impact on performance For example, a pool of SCF Proxies may be created in advance

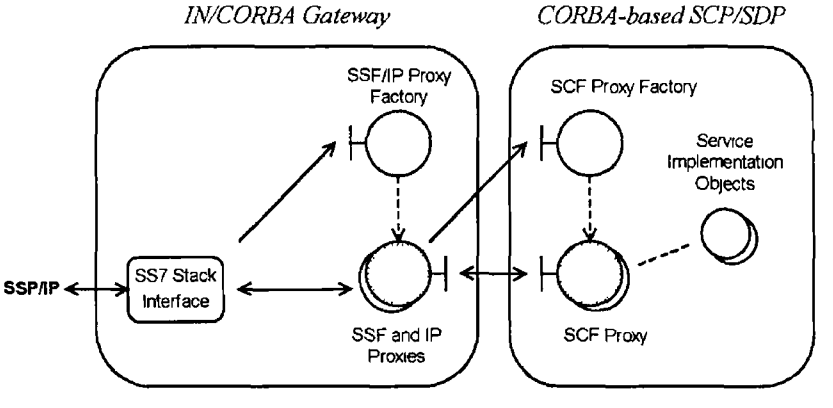


Figure 4 2 Modelling of IN/CORBA Interface Components

Considering interactions between the proxy objects, the SSF and IP Proxies accept INAP operations from the SSPs and IPs, via the SS 7 stack interface, and translate them to CORBA invocations on the SCF Proxy The SCF Proxy accepts INAP IDL invocations from other objects in the CORBA domain, transferring them to the SSF and IP Proxy objects, which translate them to the corresponding INAP operations on the SSFs and IPs It is assumed that, all of these proxy objects use the standard Q1218IN_3 DefAc IDL interface for communication, which is defined by the mapping of the ITU-T Q1218 INAP specification to IDL as per [OMG, 1999] All messages and message processing times used in the model, as

described in §4 5, have been based on this IDL specification Note that the SSF Proxy, IP Proxy and the associated functions required to interface to the SS 7 stack, are not considered to be distributable in the CORBA domain and are thus modelled as a single computational object, the *Gateway* (GW) object

4 3 2. Service Components

It is assumed that the IN service logic and data, residing on the CORBA platform, is realised by a subset of the computational objects composing the TINA Service Architecture [TINA-C, 1997]

A similar approach to that given in [Herzog & Magedanz, 1997] is adopted, which defines methods for modelling IN services executing in a TINA environment Here it is assumed that all calls originate and terminate on the IN side so that neither the calling nor called party uses a TINA end-system and thus, is not modelled as a TINA user This is appropriate for the CORBA-based SCP scenario as all SSPs resides in the IN domain and these are the only originators of calls As a result, the IN service capabilities may be encapsulated entirely within the TINA Service Session COs The TINA Access Session is not required and the COs that provide this functionality are not required

With this approach, all calls are established through the *Gateway* (GW) under the supervision of the TINA *Service Session Manager* (SSM) The service capabilities are modelled within a *User Application* (UAP), interacting with an SSM, which makes use of a service specific IN *Service Support Object* (SSO), e g a database containing number translation tables As there is no call-party specific access session, the *User Agent* (UA) is anonymous and acts on behalf of all IN users The *Provider Agent* (PA) is also generic in this case Figure 4 3 below shows the COs required, and their dependencies, for an implementation of a typical IN service (*Virtual Private Network*)

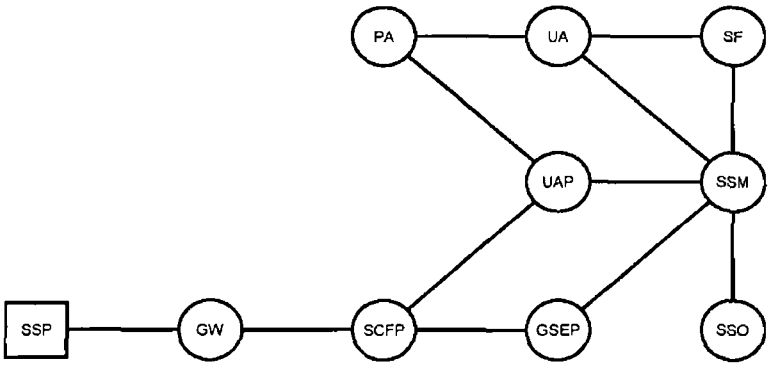


Figure 4 3 Computational Objects Required for a Typical IN Service

For any service session, on receipt of the initial service request from the SSP, the GW passes the initial call to the UAP via the *SCF Proxy* (SCFP), which in turn initiates a corresponding TINA service session via the PA The PA interacts with a UA in order to perform a generic access session for service session establishment Once the SSM has been created and initialised by the Service Factory (SF), a control relationship is established between the IN SSF and the TINA SSM The interactions between components are thence dependent on the specific service in execution Note that the GW, SCFP, PA, UA and GSEP are not specific to a particular service, whilst the SF, UAP, SSM and the SSO are all service specific

4.4. Distribution of Computational Objects

Having defined the network and the software objects being modelled, the following assumptions are made regarding the assignment of computational objects to SNs and GWSNs

- With the exception of the GW computational object, there is no restriction on assignment of computational objects to network nodes The GW may only be instantiated on the GWSNs
- COs are assumed to be atomic That is, they may not be decomposed into smaller objects and distributed between nodes

The assignment of specific COs to specific SNs and GWSNs is determined by a set of optimal methods developed in Chapter 5 The resulting optimal assignments are detailed in Chapter 6 An example of an optimal distribution for a ten-node network supporting three different services is shown in Table 4 1 below

Node	Gateway	Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x		x			x			x			
2	x	x	x	x	x		x			x			x			
3		x	x	x	x		x			x			x			
4		x	x	x	x		x	x		x	x		x	x		x
5		x	x	x	x	x		x	x		x	x		x	x	x
6		x	x	x	x	x			x			x			x	
7		x	x	x	x			x			x			x		x
8		x	x	x	x	x		x	x		x	x		x	x	x
9		x	x	x	x	x	x		x	x		x	x		x	
10		x	x	X	x		x			x			x			

Table 4 1 An Example Object Distribution

4.5. Distributed Call Model

It is assumed that all communication between COs during a service session uses the remote method call mechanism of the CORBA ORB The following sections detail the assumed behaviour and processing times associated with the ORB

4.5.1. Execution Semantics

It is assumed that an asynchronous invocation mechanism is used for all calls i e a process making a CORBA client call does not block while waiting for a response from a server (This is achievable in many commercial ORBs by use of 'call-backs' or multi-threading of client calls)

Regarding processing, execution of a call on the client side consists of processing for the appropriate client processing time, then processing for the appropriate protocol encoding time (marshalling) if the message is to be sent to a different node After these times the message has left the client node and the processor may commence processing the next message in the client-side queue On the server side, at this same instant, the message is added to the server queue for processing These processing times are detailed in the next sections It is assumed that all objects execute in one server, served by a single FIFO job queue

4 5 2 Message Processing Times

Each message passed between two COs has associated with it (i) a CORBA marshalling (protocol encoding) time on the client-side node, (ii) a CORBA de-marshalling (protocol decoding) time on the server-side node and (iii) a processing time for completion of some service specific task on both the client and server side nodes

Figure 4 4 shows execution times for messages passed between COs In the right hand figure, two COs are executing on different processors The processing times T_{CP} (client processing time) and T_{CO} (ORB marshalling time) gives the total processing time at the client node associated with this message Similarly, T_{SO} (ORB de-marshalling time) and T_{SP} give the total processing time at the server node In the left hand figure, both COs are executing on the same processor so the total processing time is given by T_{CP} and T_{SP}

Marshalling, de-marshalling and processing times are assumed to remain constant for a particular message over all sessions of a service If the communicating COs are located on the same node, the marshalling and de-marshalling times are not included in the overall processing time for the message as it is assumed that the ORB is not required for communication (i e the call is a local function call)

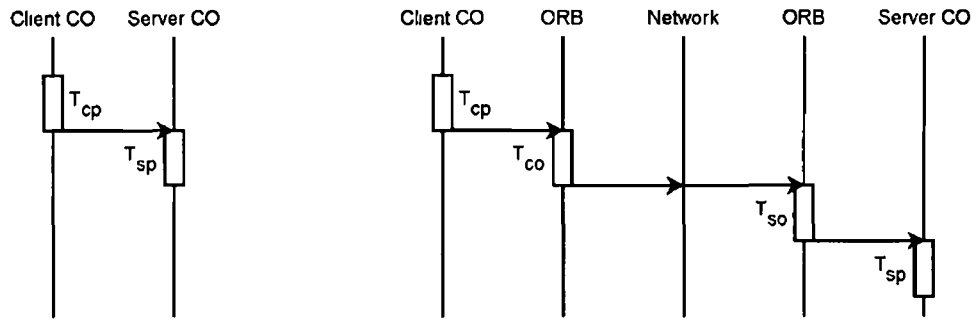


Figure 4 4 Message Processing

The marshalling and de-marshalling times used for the simulation model are derived from times we measured on a commercial ORB (Visibroker 3 3 running on a Sparc Ultra 5) The IDL used for determining timing measurements is based on the IN/CORBA specification and the TINA Retailer Reference Point specification [TINA-RET, 1999] so that each message has associated with it the appropriate marshalling and de-marshalling times Processing times for actual service related tasks are based on the processing times for the service executing on a legacy SCP These times are based on those reported in [Jennings, 1999] All processing times in the model are deterministic

4.6. Specification of Test Services

We specify three standard IN services for execution in the model

- Service A Virtual Private Network
- Service B Ringback
- Service C Restricted Access Call Forwarding

These services have varying levels of complexity Service C is the simplest with 32 messages in total and one user interaction phase Service A is more complex with 36 messages and two user interaction phases Service B is a high complexity service with 66 messages and four user interactions The following aspects of service execution are modelled

- The correct sequencing of messages during a successful session It is assumed that service users never abandon ongoing service sessions and thus it is not necessary to implement the signalling required for premature session termination Exception messages due to error conditions are also ignored
- The processing load for each message processed during a service session (as described in §4 5)
- An estimate of the delays incurred during a session due to actions on the service user (e g conversation time, digit entry) These delays are modelled as draws from a negative exponential probability distribution with an appropriate mean value

- An estimate of interaction times with the SSP (other than during user interaction) These times are similarly modelled as draws from a negative exponential probability distribution with an appropriate mean value

The objects required to execute each of the three services are shown in the Figures 4 5–4 7 below COs are shown as graph nodes and COs which interact during a service session are joined by an edge In these diagrams, components that are subscripted by the letters A, B and C are specific to Service A, Service B and Service C respectively Components that are not subscripted are used by all services

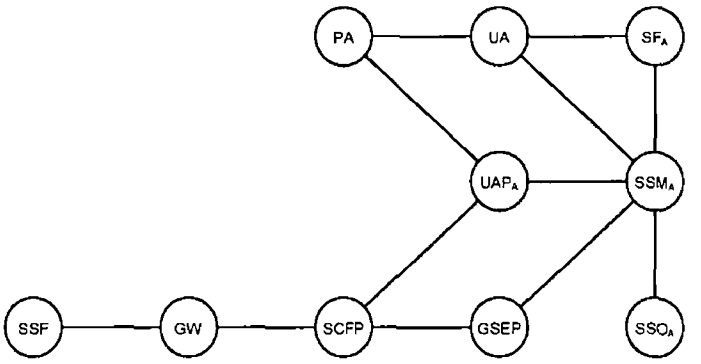


Figure 4 5 Computational Objects for Service A

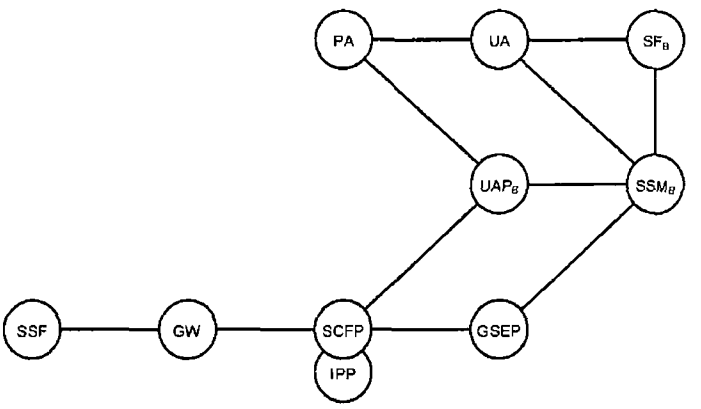


Figure 4 6 Computational Objects for Service B

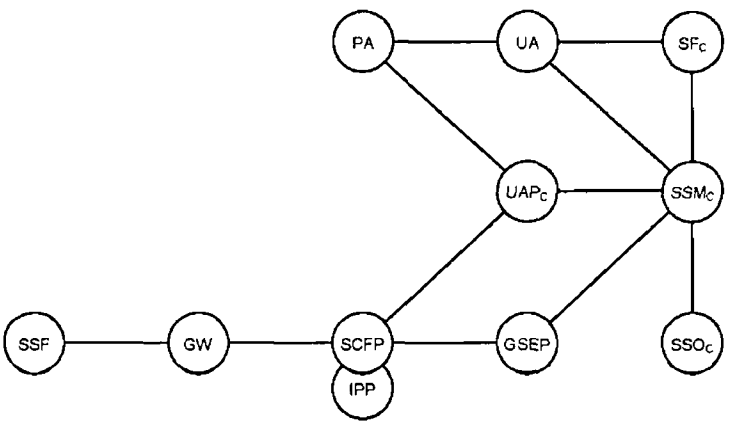


Figure 4 7 Computational Objects for Service C

4.6.1. Service A. Virtual Private Network

Service Description The *Virtual Private Network* service creates a logical sub-network spanning a single or multiple IN network domains which appears to a specific group of users as a private network, providing the types of services normally associated with private exchanges In this scenario, all calls are normally controlled by a SCP, which provides facilities such as number translation and call monitoring

Figure 4 10 (in Section 4 9) presents the message sequence chart (MSC) for a successful session of this service, as implemented using the CORBA-based SCP/SDP It also defines the user interaction phases for the session The duration of the User Interactions A1 and A2 (conversation) are to be drawn separately for each service session from a negative exponential distribution with a mean of 100 seconds

Table 4 2 (in Section 4 9) provides details of the processing times associated with each message

4 6.2. Service B Ringback

Service Description The Ringback service allows a calling party, upon receipt of an engaged tone for a specific called party, to request that a call be automatically initiated to that called party once his/her current call has terminated To realise this service in a conventional IN, the SCP signals the SSP to report when the called party's current call terminates, after which it signals the SSP to initiate a call between the calling and called parties

Figure 4 11 (in Section 4 9) presents the MSC for a successful session of this service, as implemented using the CORBA-based SCP/SDP It also defines the user interaction phases for the session Note that, for clarity, interactions with the IP are shown as using a separate IP Proxy In reality the SCF and IP Proxies are the same object

The duration of the User Interaction B1 (play announcement) is to be drawn separately for each service session from a negative exponential distribution with a mean of 5 seconds

The duration of the User Interaction B2 (conversation) is to be drawn separately for each service session from a negative exponential distribution with a mean of 100 seconds

The duration of the User Interactions B3 and B4 (phone ringing) are to be drawn separately for each service session from a negative exponential distribution with a mean of 5 seconds

The duration of the CTR message to the SSP returning with ARI to the gateway is to be drawn separately for each service session from a negative exponential distribution with a mean of 10 ms

Table 4 3 (in Section 4 9) details the processing times associated with each message

4 6.3. Service C: Restricted Access Call Forwarding

Service Description Basic call forwarding allows a user to redirect incoming calls to a different number transparently to the calling party A variation of this, in which the calling party must enter a specified PIN number before the call is forwarded to the other number is modelled here

Figure 4 12 (in Section 4 9) presents the MSC for a successful session of this service, as implemented using the CORBA-based SCP/SDP It also defines the user interaction phases for the session Note that, for clarity, interactions with the IP are shown as using a separate IP Proxy In reality the SCF and IP Proxies are the same object

The duration of the User Interactions C1 (prompt and collect user information) are to be drawn separately for each service session from a negative exponential distribution with a mean of 5 seconds

The duration of the CTR message to the SSP returning with ARI to the gateway is to be drawn separately for each service session from a negative exponential distribution with a mean of 10ms seconds

Table 4 4 (in Section 4 9) provides details of the processing times associated with each message

4 6 4 Message Details for Test Services

Details of the messages for the three test services are given in Tables 4 2-4 4 (in Section 4 9) Each message has associated with it message execution times which have been used in the simulation The message numbers match with message numbers in the MSCs (Figures 4 10–4 12, in Section 4 9) The first column of times gives the protocol encoding times in the client The second column gives protocol decoding times in the server The third column gives processing times for service related tasks in the client and the fourth gives service related processing for the server

4.6 5 Traffic and Loading Scenarios

The source of traffic for new service session initiation (*InitialDP* messages) is generated at the entry point to the GWSN node in the simulation We make the assumption that this traffic has exponentially distributed inter-arrival times and can accurately represent the amalgamated traffic from all SSPs connected to the GWSN Apart from the initial message traffic, all other messages involving interaction with the SSP or service users are modelled independently, as given in the service descriptions above The mean traffic volumes of initial requests presented to each GWSN may be varied independently according to service type The specific loading scenarios used for performance investigations are described in Chapter 6

4.7. Simulation Model Implementation

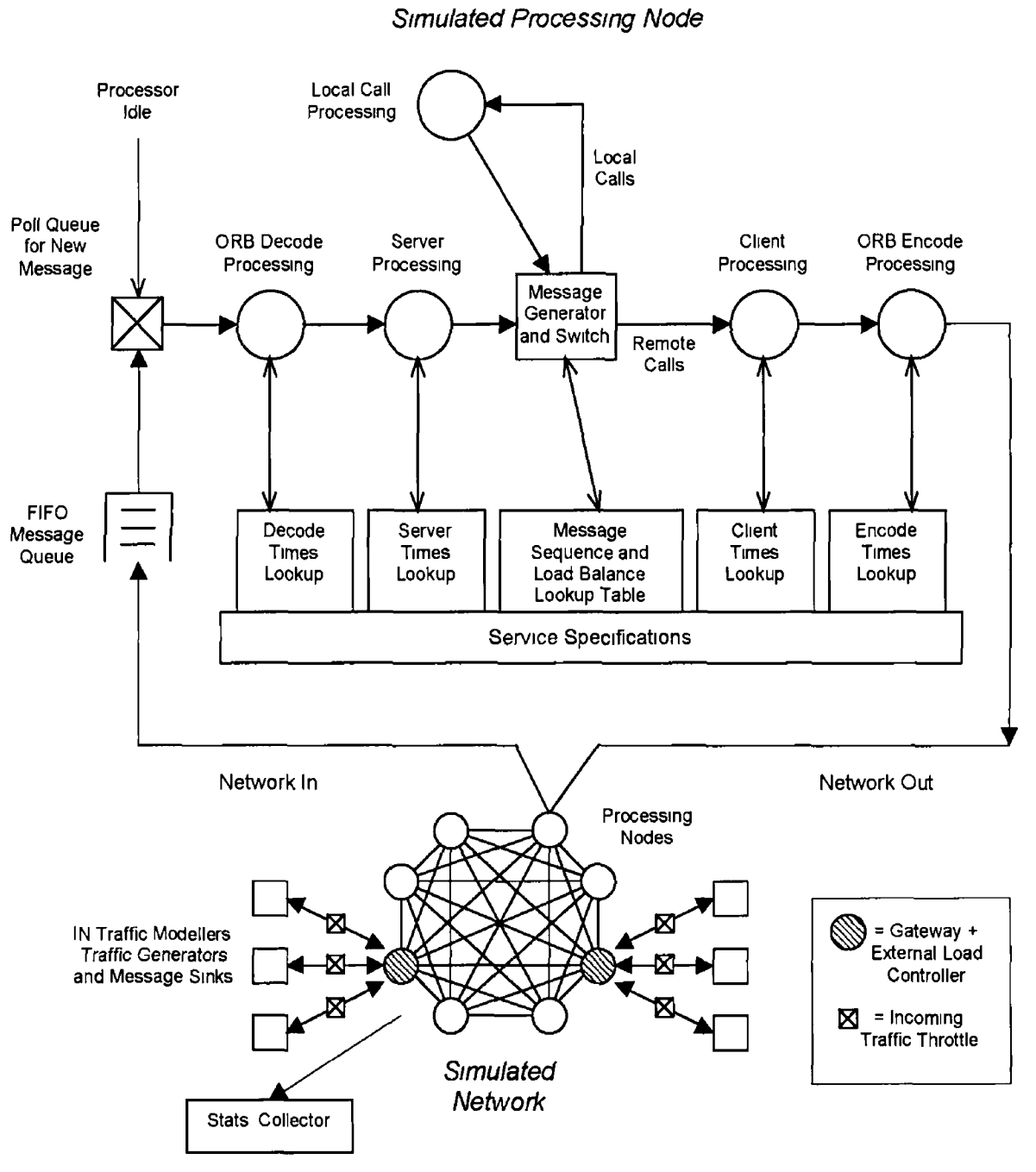
Here we describe the OPNET discrete-event simulation that was developed for performance modelling of the distributed service platform We describe in detail how the computational model maps to the simulation model and describe how performance control algorithms are implemented in the simulator

A schematic of our distributed service platform simulation model is shown in Figure 4 8 The main elements of the model are a message model, a network model and a node process model An overview of the model function is as follows Application procedure calls during service execution are represented by messages, which are routed by the network Node processes represent CORBA servers (which can also make client calls) Node processes hold messages for the appropriate processing times and also send and receive messages between each other This process behaviour represents procedure calls between COs during execution of a service As many service sessions may be in progress simultaneously, the simulation must keep track of which messages belong to which service sessions For simplicity, this is done by storing state information in the messages rather than at the nodes Traffic to and from the Intelligent Network is modelled by *IN Traffic Modellers*, which generate new service sessions (of the three different service types) The *IN Traffic Modellers* simulate processing and return of messages from/to the SSF They also collect round trip time (RTT) and rejection ratio (RR) statistics Details of each of the model elements are given below

4 7 1. Messages

Local and remote function calls between COs are represented as simple message passing in the simulator A remote call is represented as two messages - one for function invocation and one for function return Local calls are represented as a single message and remain internal to node process Messages contain fields (Figure 4 8 bottom) which are required for proper routing and processing of remote function call messages during the simulation Address fields allow function invocation/return messages to be sent and received over the network between node processes 'Type' fields identify a particular message as relating to a given function invocation/return for a given service type Session related fields store information about the particular session that the function invocation/return message relates to A time stamp field is used to calculate network delays The five message fields and their associated protocol function are as follows

(1) The *GW Address* field records the address of the source Gateway Node that initiated the service session The Traffic Generator sets this field when the first message (*InitialDP*) of a new service session is created This field is copied into all subsequent messages relating to this session This information allows messages to be sent back to the correct service user (*IN Traffic Modeller*)



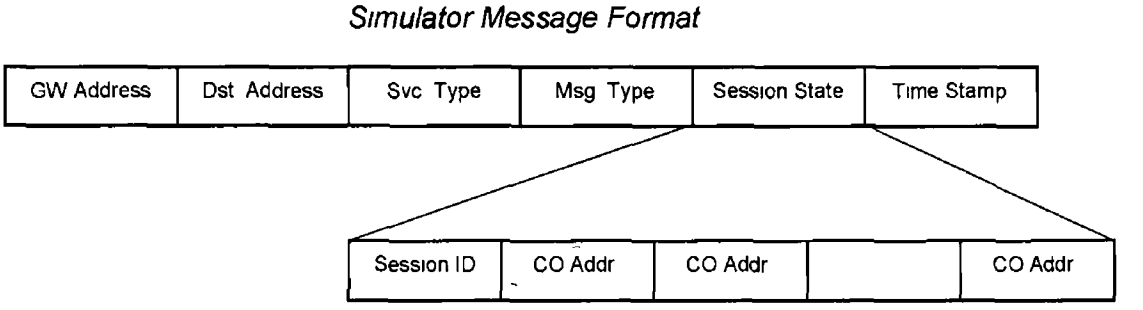


Figure 4 8 Service Platform Simulator

(2) The *Destination Address* allows basic routing of messages from node to node and back and forth from the *IN Traffic Modeller* This address is initially set in the traffic generator to be the address of the processing node to which it connects Subsequently, it is modified in node processes for re-routing of the message Node processes may also create new messages (at forking points in the MSC) and must set the destination address appropriately The destination that is set or modified in node processes will be determined by the internal performance controller (load balancing algorithm) and the session state information (see below) The load-balancing scheme has knowledge of placement of all COs in the network and so will only route a message towards a node that hosts the target CO

(3) The service type (*Svc Type*) and (4) message type (*Msg Type*) fields simply identify which function invocation/return of a particular service the message represents This information is required by the node process so that the correct message encoding/decoding and processing times may be applied These times are as per the message details tables given in §4 9

(5) *Session State* field To reduce the complexity of the node process, messages store service session state information so that the node process does not need to hold state information on all sessions currently active (This is not intended to represent how a distributed system would function in reality but it is functionally equivalent to storing state in the process and will have no impact on simulated performance) Any new messages, created in the context of a particular session, copy all information in the *Session State* field from the previous message This preserves the state across all messages of the session

The Session State field is expanded in Figure 4 8 (bottom) The *Session ID* sub-field records the particular session to which the message belongs This ID is set by the Traffic Generator and incremented for every new session created This allows all active sessions to be correctly tallied The Session ID is primarily used, in conjunction with the *Time Stamp* field, to calculate service session completion times (RTTs)

The *Session ID* field also contains a list of CO addresses (the *CO Addr* sub-fields) The purpose of this is to maintain correct session execution semantics That is, once a particular CO instance (at a particular node) has been invoked during a service session, it must be used for the remainder of that session (Otherwise the implication is that, in a real distributed system, different CO instances would be able to somehow share state during a service session - which isn't normally the case) Each time a CO type is called for the first time during a service session, a *CO Addr* field is added to the list and set with the ID of the CO (which is known system wide) and the destination address of the node hosting the CO This information is then searched if the same CO type is required later in the session If there is no record of its previous use, then the *load balancing lookup table* is used to select a node that hosts the CO

(6) The *Time Stamp* field is set by the Traffic Generator when the first message in the session is created This time stamp is copied to all subsequent messages in that session This time stamp may be subtracted from the service session ending time in order to compute the RTT for the session

As service sessions do not end at the *IN Traffic Modeller*, but instead internally to the service platform, a special message is sent from the network to the traffic modeller after processing of the last service message has finished This message has zero processing times and reaches the modeller in zero simulation time This allows all RTT data to be collated at the traffic modeller

4.7.2. The Network

The network's function is simply to route messages between nodes without delays As discussed earlier, it is assumed that protocol encoding/decoding times are considerably larger than network transmission times We assume that the network is over-capacitated so that even when nodes are under heavy load, network processors and queues are still lightly loaded and do not contribute significant delay

4.7.3. Processes

Each processing node in the network has an identical processor model as shown in Figure 4 8 (top) The main elements of the node model are a FIFO input message queue, a queue polling mechanism, a number of processing stages, a message generator and a set of service specifications, one for each service type

Messages arriving from the network are queued until the queue is polled When polled, the message at the head of the queue is removed and processed in the *ORB Decode Processor* The message is held for the appropriate de-marshalling time, which is specific to the message type (identified by the *Msg Type* and *Svc Type* message fields) The message is then passed on for a message specific server-side processing holding time in the *Server Processor* At the end of processing the message is evaluated by the *Message Generator and Switch* This evaluation is based on a combination of the appropriate MSC for the particular message, the *Session State*, and the load balance lookup values, as described below

Firstly, the next message(s) in the message sequence and the corresponding target CO(s) for the message are looked up If there is a fork at this point in the MSC, more than one new message/target-CO pair are generated and these are internally queued in the *Message Generator and Switch* The target CO for each new message is then evaluated in terms of the *Session State* information This may determine that a message should be (a) processed locally (if the target CO has been called previously in this session and resides locally) or (b)

processed at a particular remote node (if the target CO has been called previously in this session and resides remotely) If there is no information on the session state of the target CO, then the load balancing table is looked up to choose a destination node This may in turn result in (a) or (b) For (a) the message is processed locally and the processing time is the sum of the server and client processing times for the message type, as illustrated in Figure 4 4 earlier After local processing the message is returned to the *Message Generator and Switch* for further evaluation and retrieval of the next message/target-CO This may lead to a cycle of local message processing until the end of the service or until a remote call is required For (b) the outgoing message then receives appropriate client-processing and ORB-encoding processing and is then sent out to the network for routing to its target CO

Note that the processing stages (circles) in Figure 4 8 together represent a single process That is, only one may be processing at any given time Thus, a processing cycle initiated by an incoming message will continue without interruption until all messages generated by the incoming message are processed and exit the node or until the last message of the service is finished execution At this point the input queue is polled for a new message and the cycle repeats (if the queue is not empty) This condition is signalled to the polling switch as soon as the process (all processing stages) becomes idle (We assume there is no delay due to the polling mechanism)

Note that although outgoing messages represent remote method calls, we assume that these calls are fully asynchronous (i e they do not put the process into a blocked state whilst waiting for return messages for the call)

4 7 4 IN Traffic Modellers

The *IN Traffic Modellers* both create initial service requests from the SSF and respond to messages sent to the SSF Thus they model the IN side of the network As mentioned previously, traffic for new service requests (the *InitialDP* INAP messages) has exponentially distributed inter-arrival times Traffic sources for each service and from each traffic generator are independent (i e there are six independent Poisson sources in the simulation) The mean rate of traffic generation can be varied independently for each service type and each traffic source node When the Traffic Modeller receives a request destined for the SSF, it starts a timer, to model the length of the interaction with the SSF This may be a user interaction time or solely a processing time at the SSF These times are also exponentially distributed with mean values as stated in §4 6

Gateway Nodes

As shown in Figure 4 8, the *IN Traffic Modellers* each connect to a single node, the gateway node (GWSN) The gateway nodes are the only nodes to host the *Gateway* CO (GW) but are

otherwise normal processing nodes, that may host any other computational objects The gateway nodes also host the *External Load Controller* (described below)

Performance Data Collection

Several statistics are collected periodically by the *Stats Collectors*

Load Each node process keeps a running account of the amount of time it is busy processing messages over a given statistics collection time interval At the end of this time interval the processor load is calculated as the busy time divided by the length of the statistics collection interval and is sent to the *Stats Collector* This process is repeated for each consecutive interval

Rejection Ratios The *IN Traffic Modellers* keep a count of all service session initiation messages (*InitialDPs*) that are accepted and rejected by the *External Load Controllers* over the statistics collection interval The *rejection ratio* is calculated at the end of the k^{th} interval as

$$R_i(k) = \frac{rejects_i(k)}{rejects_i(k) + accepts_i(k)}$$

where $rejects_i(k)$ and $accepts_i(k)$ are the number of rejected and accepted sessions respectively at the i^{th} gateway during the k^{th} interval

Round Trip Time The RTT is the time from session start to session end and is calculated at the *IN Traffic Modeller* at the end of each service session The RRT for each session is sent to the *Stats Collector* along with the relevant session information (the service type and originating gateway address)

4 7.5 Operation of Simulated Performance Controls

Both *external* and *internal* performance controls are implemented in the simulation These operate by use of lookup tables, which are updated periodically by the performance control algorithms Different algorithms may be plugged into the simulation

The internal performance control (load balancing) is implemented by periodically updating the *Load Balance Lookup Table*, which is referenced by the *Message Generator and Switch* This lookup table contains a list of target COs each with a list of their corresponding host nodes and *splitting probabilities* Entries in the table thus have the following form

```
CO_ID   {(NODE_ID_1 P_SPLIT_1) (NODE_ID_2 P_SPLIT_2)      (NODE_ID_N P_SPLIT_N)     }
```

A node from this list is chosen according to a *Bernoulli Trial* where the probability of choosing NODE_ID_N is given by P_SPLIT_N For each CO entry the sum of all splitting

probabilities must be equal to one This method allows any of the internal performance controls investigated in this thesis to be 'plugged-in' to the simulator

The *external performance control* requires that offered traffic to the network is limited during periods of high load The *External Load Controllers* at the gateway nodes implement this function by considering the load values collected by the *Stats Collectors* To justify our simulation of our external controls, we consider how the *External Load Controller* could interact with the *IN Overload Control*

We assume that the IN Overload Detection (§2 4 1) is implement in the SS 7 stack processes, which are locally accessible to the *Gateway Object* The IN Overload Detection and the *External Load Controller* may thus communicate directly Rather than detecting load independently, the IN Overload Detection module is given the current highest expected load value of the processors in the CORBA service platform by the *External Load Controller* This value is resent at the end of each control interval, which is assumed to be at least as long as the statistics collection period This average load value is taken as being equivalent to an SCP processor utilisation value in a normal Intelligent Network and the Overload Detection module signals the SSPs to throttle according to the assumed SCP utilisation The function of the *External Load Controller* would be simply to detect the highest processor load on the CORBA service platform and pass it to the IN Overload Detection module

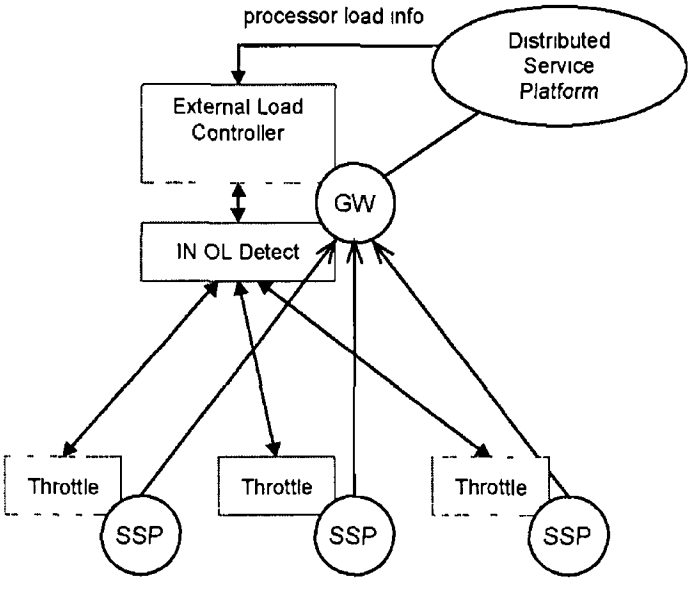


Figure 4 9 Assumed External Controller Interaction with IN Overload Control

Note that we do not explicitly model the SSPs or the IN Overload Control algorithms In our simulation we assume that the IN Overload Control will act instantaneously to throttle (or call gap) traffic from the SSP at the required rate We assume that this can be modelled by

throttling input traffic between the traffic generator and the gateways This function is carried out by the *incoming traffic throttles* (Figure 4 8) which are controlled by the particular external performance control in operation in the simulation

4.8. Chapter Summary

The chapter has described the computational object model and simulation model of the distributed service platform under study in this thesis New strategies for CO distribution on the platform and external and internal performance controls are proposed in the next chapter and in Chapter 6 we use the simulation model, verified by an analytic model, to investigate performance

4.9. Chapter 4 Appendix – Test Service Details

(Specification of message sequences and message processing details for simulation test services)

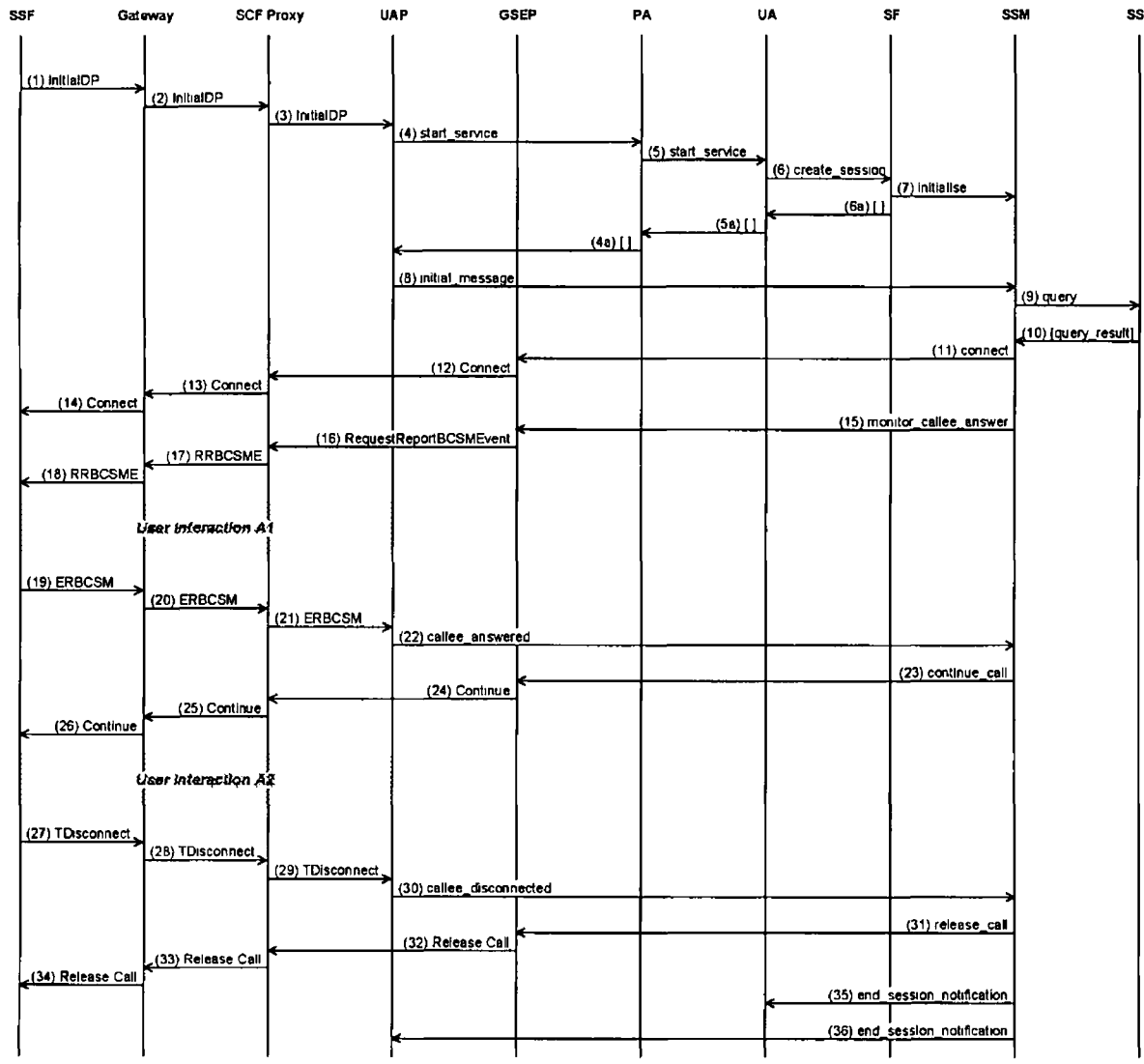


Figure 4 10 MSC for Service A

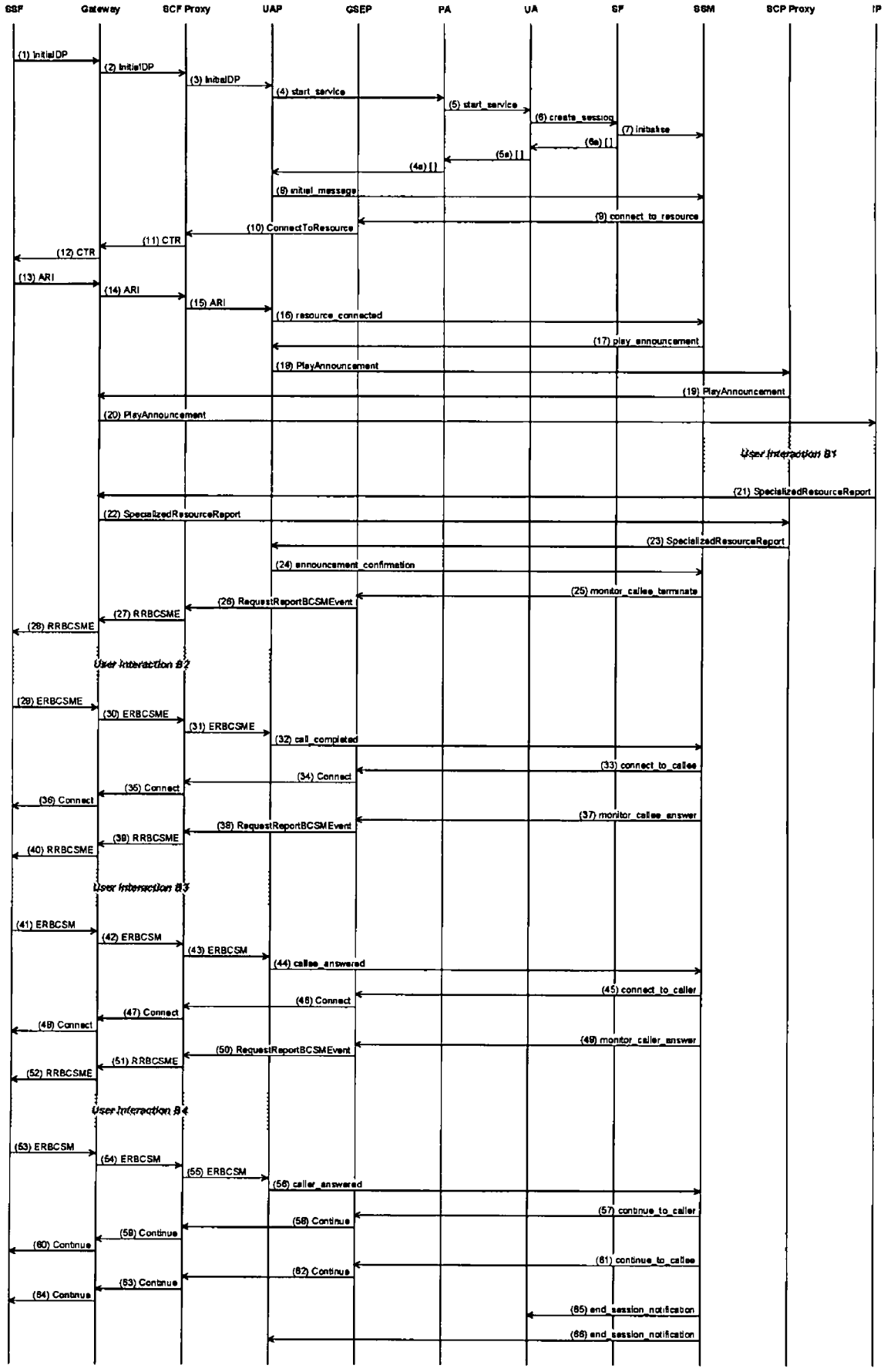


Figure 4 11 MSC for Service B

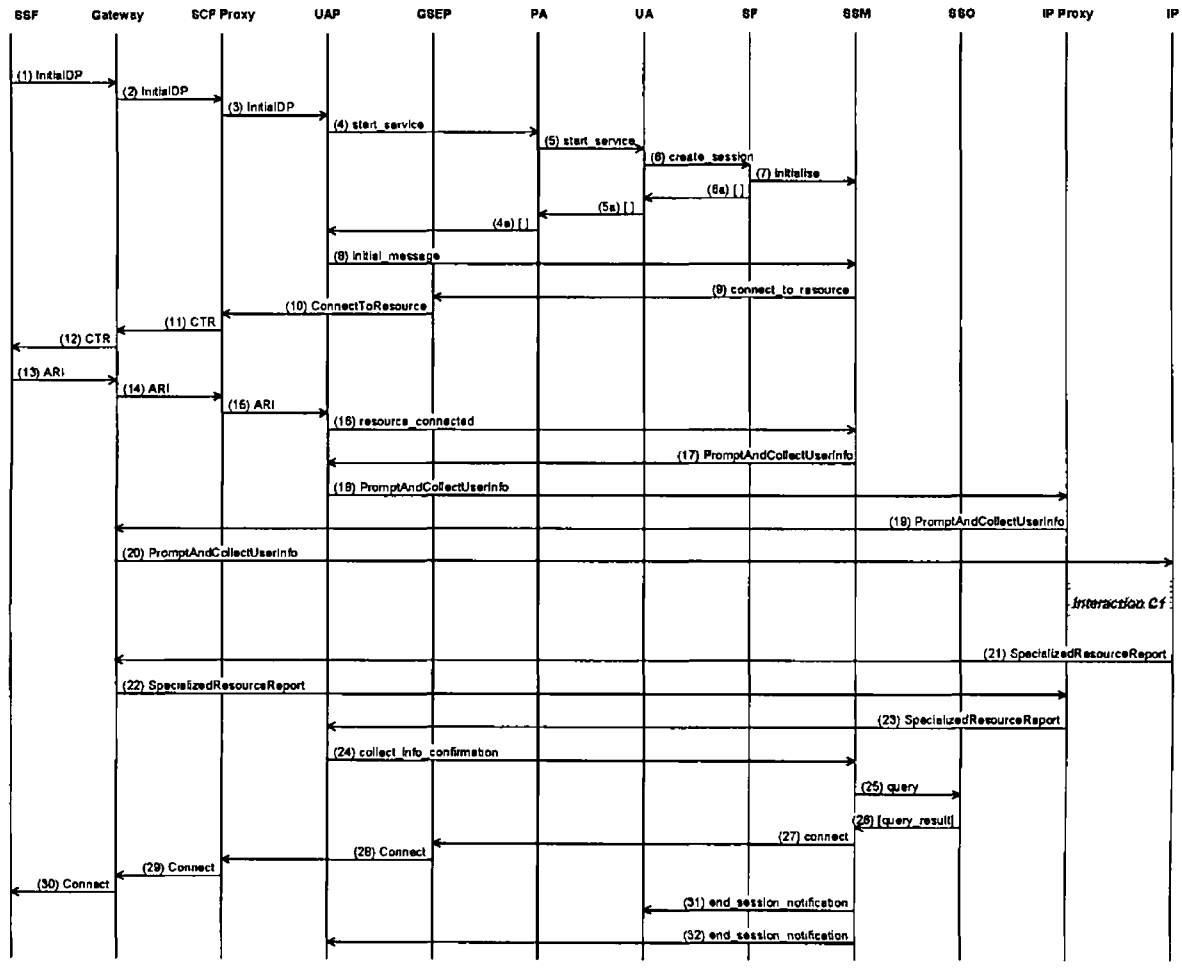


Figure 4 12 MSC for Service C

Service A (VPN) Message Details					Processing Time ORB (ms)		Processing Time Service (ms)	
Message #	Client Object	Server Object	Interface	Message	Client	Server	Client	Server
1	SSF	Gateway	TCAP	InitialDP	NA	NA	NA	1 00
2	Gateway	SCF Proxy	Q1218_3 DefAc	InitalDP	0 50	1 50	0 25	0 50
3	SCF Proxy	UAP	Q1218_3 DefAc	InitialDP	2 00	1 50	0 50	0 50
4	UAP	PA	I_paUAP	start_service	2 00	1 50	0 50	0 50
5	PA	UA	I_uaAccess	start_service	2 00	1 50	0 50	0 50
6	UA	SF	I_sfCreate	create_session	2 50	2 00	0 50	2 00
7	SF	SSM	I_ssmInitialise	initialise	2 50	2 00	0 50	2 00
6a	SF	UA	I_sfCreate	[create_session return]	2 50	2 00	0 50	1 00
5a	UA	PA	I_uaAccess	[start_service return]	2 50	2 00	0 50	1 00
4a	PA	UAP	I_paUAP	[start_service return]	1 50	2 00	0 50	1 00
8	UAP	SSM	not defined	initial_message	2 00	1 50	0 50	3 00
9	SSM	SSO	not defined	query	1 50	1 00	2 50	5 50
10	SSO	SSM	not defined	query_result	1 00	1 50	3 50	3 50
11	SSM	GSEP	not defined	connect	1 50	1 00	0 50	0 50
12	GSEP	SCF Proxy	Q1218_3 DefAc	Connect	2 00	1 50	0 50	0 50
13	SCF Proxy	Gateway	Q1218_3 DefAc	Connect	2 00	0 75	0 50	0 25
14	Gateway	SSF	TCAP	Connect	NA	NA	0 50	NA
15	SSM	GSEP	not defined	monitor_callee_answer	1 50	1 00	0 50	1 00
16	GSEP	SCF Proxy	Q1218_3 DefAc	RequestReportBCSMEvent	2 00	1 50	0 50	0 50
17	SCF Proxy	Gateway	Q1218_3 DefAc	RequestReportBCSMEvent	2 00	0 50	0 50	0 25
18	Gateway	SSF	TCAP	RequestReportBCSMEvent	NA	NA	0 50	NA
19	SSF	Gateway	TCAP	EventReportBCSM	NA	NA	NA	0 50
20	Gateway	SCF Proxy	Q1218_3 DefAc	EventReportBCSM	0 50	1 50	0 25	0 50
21	SCF Proxy	UAP	Q1218_3 DefAc	EventReportBCSM	2 00	1 50	0 50	0 50
22	UAP	SSM	not defined	callee_answered	1 50	1 00	0 50	1 00
23	SSM	GSEP	not defined	continue_call	1 50	1 00	0 50	0 50
24	GSEP	SCF Proxy	Q1218_3 DefAc	Continue	1 50	1 00	0 50	0 50
25	SCF Proxy	Gateway	Q1218_3 DefAc	Continue	1 50	0 50	0 50	0 25
26	Gateway	SSF	TCAP	Continue	NA	NA	0 50	NA
27	SSF	Gateway	TCAP	TDisconnect	NA	NA	NA	0 50
28	Gateway	SCF Proxy	Q1218_3 DefAc	TDisconnect	0 50	1 50	0 25	0 50
29	SCF Proxy	UAP	Q1218_3 DefAc	TDisconnect	2 00	1 50	0 50	0 50
30	UAP	SSM	not defined	callee_disconnected	1 50	1 00	0 50	1 00
31	SSM	GSEP	not defined	release_call	1 50	1 00	0 50	0 50
32	GSEP	SCF Proxy	Q1218_3 DefAc	ReleaseCall	1 50	1 00	0 50	0 50
33	SCF Proxy	Gateway	Q1218_3 DefAc	ReleaseCall	1 50	0 50	0 50	0 25
34	Gateway	SSF	TCAP	ReleaseCall	NA	NA	0 50	NA
35	SSM	UA	I_usSMEvent	end_session_notification	1 50	1 00	0 50	0 50
36	SSM	UAP	I_UAPEvent	end_session_notification	1 50	1 00	0 50	0 50

Table 4 2 Message Details for Service A

Service B (Ring Back) - Message Details					Processing Time ORB (ms)		Processing Time Service (ms)	
Message #	Client Object	Server Object	Interface	Message	Client	Server	Client	Server
1	SSF	Gateway	TCAP	InitialDP	NA	NA	NA	1.00
2	Gateway	SCF Proxy	Q1218_3::DefAc	InitialDP	1.00	1.50	0.25	0.50
3	SCF Proxy	UAP	Q1218_3::DefAc	InitialDP	2.00	1.50	0.50	0.50
4	UAP	PA	i_paUAP	start_service	2.00	1.50	0.50	0.50
5	PA	UA	i_uaAccess	start_service	2.00	1.50	0.50	0.50
6	UA	SF	I_sfCreate	create_session	2.50	2.00	0.50	1.00
7	SF	SSM	I_ssmInitiaise	initialise	2.50	2.00	0.50	1.00
6a	SF	UA	i_sfCreate	[create_session return]	1.50	2.00	0.50	0.50
5a	UA	PA	i_uaAccess	[start_service return]	1.50	2.00	0.50	0.50
4a	PA	UAP	i_paUAP	[start_service return]	1.50	2.00	0.50	0.50
8	UAP	SSM	not defined	initial_message	2.00	1.50	0.50	1.00
9	SSM	GSEP	not defined	connect_to_resource	1.50	1.00	0.50	1.00
10	GSEP	SCF Proxy	Q1218_3::DefAc	ConnectToResource	2.00	1.50	0.50	0.50
11	SCF Proxy	Gateway	Q1218_3::DefAc	ConnectToResource	2.00	0.50	0.50	0.25
12	Gateway	SSF	TCAP	ConnectToResource	NA	NA	0.50	NA
13	SSF	Gateway	TCAP	AssistRequestInstructions	NA	NA	NA	0.50
14	Gateway	SCF Proxy	Q1218_3::DefAc	AssistRequestInstructions	0.50	1.50	0.25	0.50
15	SCF Proxy	UAP	Q1218_3::DefAc	AssistRequestInstructions	2.00	1.50	0.50	0.50
16	UAP	SSM	not defined	resource_connected	1.50	1.00	0.50	1.00
17	SSM	UAP	not defined	play_announcement	1.50	1.00	0.50	0.50
18	UAP	IP Proxy	Q1218_3::DefAc	PlayAnnouncement	2.00	1.50	0.50	0.50
19	IP Proxy	Gateway	Q1218_3::DefAc	PlayAnnouncement	2.00	0.50	0.50	0.25
20	Gateway	IP	TCAP	PlayAnnouncement	NA	NA	0.50	NA
21	IP	Gateway	TCAP	SpecializedResourceReport	NA	NA	NA	0.50
22	Gateway	IP Proxy	Q1218_3::DefAc	SpecializedResourceReport	0.50	1.00	0.25	0.50
23	IP Proxy	UAP	Q1218_3::DefAc	SpecializedResourceReport	1.50	1.00	0.50	0.50
24	UAP	SSM	not defined	announcement_confirmation	1.50	1.00	0.50	0.50
25	SSM	GSEP	not defined	monitor_callee_terminate	1.50	1.00	0.50	1.00
26	GSEP	SCF Proxy	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	1.50	0.50	0.50
27	SCF Proxy	Gateway	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	0.50	0.50	0.25
28	Gateway	SSF	TCAP	RequestReportBCSMEvent	NA	NA	0.50	NA
29	SSF	Gateway	TCAP	EventReportBCSM	NA	NA	NA	0.50
30	Gateway	SCF Proxy	Q1218_3::DefAc	EventReportBCSM	0.50	1.50	0.25	0.50
31	SCF Proxy	UAP	Q1218_3::DefAc	EventReportBCSM	2.00	150.00	50.00	50.00
32	UAP	SSM	not defined	call_completed	1.50	100.00	50.00	100.00
33	SSM	GSEP	not defined	connect_to_callee	1.50	1.00	0.50	0.50
34	GSEP	SCF Proxy	Q1218_3::DefAc	Connect	2.00	1.50	0.50	0.50
35	SCF Proxy	Gateway	Q1218_3::DefAc	Connect	2.00	0.50	0.50	0.25
36	Gateway	SSF	TCAP	Connect	NA	NA	0.50	NA
37	SSM	GSEP	not defined	monitor_callee_answer	1.50	1.00	0.50	0.50
38	GSEP	SCF Proxy	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	1.50	0.50	0.50
39	SCF Proxy	Gateway	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	0.50	0.50	0.25
40	Gateway	SSF	TCAP	RequestReportBCSMEvent	NA	NA	0.50	NA
41	SSF	Gateway	TCAP	EventReportBCSM	NA	NA	NA	0.50
42	Gateway	SCF Proxy	Q1218_3::DefAc	EventReportBCSM	0.50	1.50	0.25	0.50
43	SCF Proxy	UAP	Q1218_3::DefAc	EventReportBCSM	2.00	1.50	0.50	0.50
44	UAP	SSM	not defined	callee_answered	1.50	1.00	0.50	1.00
45	SSM	GSEP	not defined	connect_to_caller	1.50	1.00	0.50	0.50
46	GSEP	SCF Proxy	Q1218_3::DefAc	Connect	2.00	1.50	0.50	0.50
47	SCF Proxy	Gateway	Q1218_3::DefAc	Connect	2.00	0.50	0.50	0.25
48	Gateway	SSF	TCAP	Connect	NA	NA	0.50	NA
49	SSM	GSEP	not defined	monitor_caller_answer	1.50	1.00	0.50	0.50
50	GSEP	SCF Proxy	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	1.50	0.50	0.50
51	SCF Proxy	Gateway	Q1218_3::DefAc	RequestReportBCSMEvent	2.00	0.50	0.50	0.25
52	Gateway	SSF	TCAP	RequestReportBCSMEvent	NA	NA	0.50	NA
53	SSF	Gateway	TCAP	EventReportBCSM	NA	NA	NA	0.50
54	Gateway	SCF Proxy	Q1218_3::DefAc	EventReportBCSM	0.50	1.50	0.25	0.50
55	SCF Proxy	UAP	Q1218_3::DefAc	EventReportBCSM	2.00	1.50	0.50	0.50
56	UAP	SSM	not defined	caller_answered	1.50	1.00	0.50	1.00
57	SSM	GSEP	not defined	continue_to_caller	1.50	1.00	0.50	0.50
58	GSEP	SCF Proxy	Q1218_3::DefAc	Continue	1.50	1.00	0.50	0.50
59	SCF Proxy	Gateway	Q1218_3::DefAc	Continue	1.50	0.50	0.50	0.25
60	Gateway	SSF	TCAP	Continue	NA	NA	0.50	NA
61	SSM	GSEP	not defined	continue_to_callee	1.50	1.00	0.50	0.50
62	GSEP	SCF Proxy	Q1218_3::DefAc	Continue	2.00	0.50	0.50	0.50
63	SCF Proxy	Gateway	Q1218_3::DefAc	Continue	2.00	0.50	0.50	0.25
64	Gateway	SSF	TCAP	Continue	NA	NA	0.50	NA
65	SSM	UA	I_usSMEvent	end_session_notification	1.50	1.00	0.50	0.50
66	SSM	UAP	I_UAPEvent	end_session_notification	1.50	1.00	0.50	0.50

Table 4.3: Message Details for Service B

Service C (Restricted Access Call Forwarding) Message Details					Processing Time ORB (ms)		Processing Time Service (ms)	
Message #	Client Object	Server Object	Interface	Message	Client	Server	Client	Server
1	SSF	Gateway	TCAP	InitialDP	NA	NA	NA	1 00
2	Gateway	SCF Proxy	Q1218_3 DefAc	InitialDP	0 50	1 50	0 25	0 50
3	SCF Proxy	UAP	Q1218_3 DefAc	InitialDP	2 00	1 50	0 50	0 50
4	UAP	PA	I_paUAP	start_service	2 00	1 50	0 50	0 50
5	PA	UA	I_uaAccess	start_service	2 00	1 50	0 50	0 50
6	UA	SF	I_sfCreate	create_session	2 50	2 00	0 50	1 00
7	SF	SSM	I_ssmInitiaise	initialise	2 50	2 00	0 50	1 00
6a	SF	UA	I_sfCreate	[create_session return]	1 50	2 00	0 50	0 50
5a	UA	PA	I_uaAccess	[start_service return]	1 50	2 00	0 50	0 50
4a	PA	UAP	I_paUAP	[start_service return]	1 50	2 00	0 50	0 50
8	UAP	SSM	not defined	initial_message	2 00	1 50	0 50	0 50
9	SSM	GSEP	not defined	connect_to_resource	1 50	1 00	0 50	1 00
10	GSEP	SCF Proxy	Q1218_3 DefAc	ConnectToResource	2 00	1 50	0 50	0 50
11	SCF Proxy	Gateway	Q1218_3 DefAc	ConnectToResource	2 00	0 50	0 50	0 25
12	Gateway	SSF	TCAP	ConnectToResource	NA	NA	0 50	NA
13	SSF	Gateway	TCAP	AssistRequestInstructions	NA	NA	NA	0 50
14	Gateway	SCF Proxy	Q1218_3 DefAc	AssistRequestInstructions	0 50	1 50	0 25	0 50
15	SCF Proxy	UAP	Q1218_3 DefAc	AssistRequestInstructions	2 00	1 50	0 50	0 50
16	UAP	SSM	not defined	resource_connected	1 50	1 00	0 50	0 50
17	SSM	UAP	not defined	PromptAndCollectUserInfo	1 50	1 00	0 50	0 50
18	UAP	IP Proxy	Q1218_3 DefAc	PromptAndCollectUserInfo	2 00	1 50	0 50	0 50
19	IP Proxy	Gateway	Q1218_3 DefAc	PromptAndCollectUserInfo	2 00	0 50	0 50	0 25
20	Gateway	IP	TCAP	PromptAndCollectUserInfo	NA	NA	0 50	NA
21	IP	Gateway	TCAP	SpecializedResourceReport	NA	NA	NA	0 50
22	Gateway	IP Proxy	Q1218_3 DefAc	SpecializedResourceReport	0 50	1 50	0 25	0 50
23	IP Proxy	UAP	Q1218_3 DefAc	SpecializedResourceReport	2 00	1 50	0 50	0 50
24	UAP	SSM	not defined	collect_info_confirmation	1 50	1 00	0 50	0 50
25	SSM	SSO	not defined	query	1 50	1 00	2 50	5 50
26	SSO	SSM	not defined	query_result	1 00	1 50	3 50	3 50
27	SSM	GSEP	not defined	connect	1 50	1 00	0 50	0 50
28	GSEP	SCF Proxy	Q1218_3 DefAc	Connect	2 00	1 50	0 50	0 50
29	SCF Proxy	Gateway	Q1218_3 DefAc	Connect	2 00	0 50	0 50	0 25
30	Gateway	SSF	TCAP	Connect	NA	NA	0 50	NA
31	SSM	UA	I_usSMEvent	end_session_notification	1 50	1 00	0 50	0 50
32	SSM	UAP	I_UAPEvent	end_session_notification	1 50	1 00	0 50	0 50

Table 4 4 Message Details for Service C

Chapter 5. Computational Object Allocation and Performance Control Strategies

This chapter presents a method for optimising the placement of software objects on network nodes, for multi-service distributed application networks Also presented are an optimal method for load distribution and load throttling in these networks (§5 1 6) and a sub-optimal market-based solution to the same problems (§5 2)

5.1. Optimal Algorithms for Object Distribution and Load Control

This section presents a new method for optimising the placement of software objects on network nodes, for multi-service distributed applications Also addressed are the related problems of optimal distribution of traffic between distributed software object instances and optimal admission control for the network The object placement problem is formulated as a *Mixed Integer Programming* (MIP) flow problem The solution yields the placement of application objects that gives the maximum allowable arrival intensities to the network under the constraints of processor load limits and object installation costs Given the optimal object placement, a further method is developed for optimising routing between object instances to maximise network revenue when arrival intensities vary over time from the original design point This problem is formulated as a *Linear Programming* (LP) problem that is constrained by the solution of the original MIP problem The solution gives the basis for a load distribution and load control algorithm for the optimised network

5 1.1. Strategy Overview

The model under study consists of a network of fully-connected processors of non-uniform capabilities and processing capacities, serving multiple customer classes Service execution in the network consists of message passing between instances of Computational Objects residing on network processors Flow costs in the network are derived from processing and protocol encoding/decoding times for these service messages Processing times for objects are allowed to vary across processors so that multi-processor nodes may be included in the model

The placement of objects on processors in the network, is a critical distributed system design decision that determines the maximum service rate of the network as a whole This is especially true when a significant amount of processing is required to distribute messages (Protocol encoding/decoding times have been compared to service processing times for telecom services, executing on a distributed platform, in [McArdle *et al*, 2000]) The object placement problem has been undertaken in [Anagnostou, 2000] where the total communication cost in the network is minimised given the set of flows between all object instances and given the service demand volumes from users This problem is reformulated (in §5 1 4) so that the total allowable user demand is maximised given the relative volumes of requests from users Also, to limit the costs of replicating objects on processors, linear installation costs are added to the problem (§5 1 5) This formulation effectively maximises network throughput and has the advantage of balancing load between processors which in turn ensures the maximum amount of spare processing capacity is available when the network is under-loaded (A simple optimisation model for TINA service components, also focusing on throughput maximisation, has been presented in [Kihl, 1997] In this work, it was assumed that component copies share traffic evenly This is not assumed here Also, protocol encoding/decoding times are not taken into account in [Kihl, 1997]) As this approach determines the maximum allowable demand, it is suitable as the basis of an admission control strategy Such a strategy is given in §5 1 6 This is then extended to revenue maximisation admission control in Section §5 1 7 Revenue maximisation for Intelligent Networks has been studied in [Lodge, 1999] A similar objective is adopted here for use with the distributed system model However, a fairness constraint is added to the revenue problem to allow the bias towards high revenue customers to be damped at the cost of lower system revenues

The following section describes the model formally and introduces some notation for the network and service topologies This model is used in the subsequent sections for defining the object placement, random splitting and admission control optimisation problems

5 1 2 Model Notation

The system model consists of a set of heterogeneous processors N Each processor is connected to all other processors in the system by a network A set of processors $N_{user} \subseteq N$ has connections to a set of system *users*, which generate new service requests to the system

Each service request to the system belongs to the set of service types S Requests of type $s \in S$ require a sequences of messages Q_s to be passed between processors, and between processors and the user in order to complete service The execution of Q_s is referred to as a *service session* of type s The set of distinct messages in the sequence Q_s is denoted M^s The set of all messages is denoted $M = \cup M^s$ over $\forall s \in S$

5 1 2 1 Messages and Computational Objects

The set of all *computational objects* C is defined next Each computational object $c_i \in C$ is defined as the capability to send the set of messages $M_i^{client} \subset M$ and to receive the set of messages $M_i^{server} \subset M$, where $M_i^{client} \cap M_i^{server} = \varnothing$

The set of processors that *support* the computational object $c_i \in C$ is denoted $N_i \subseteq N$ That is, if $n \in N_i$, it is said that an *instance* of computational object c_i may be allocated to processor n Note that this implies that a processor n may send a message $k \in M_i^{client}$ if and only if $n \in N_i$ and a processor m may receive a message $k \in M_j^{server}$ if and only if $m \in N_j$

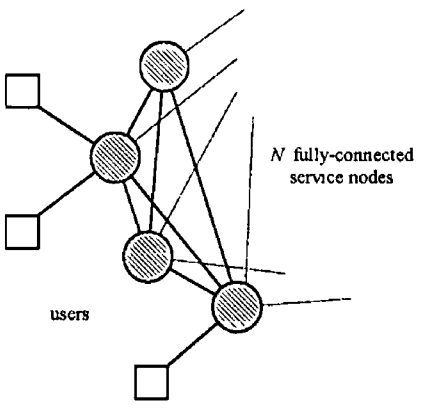


Figure 5 1 Network Model

The set of all objects required to execute a service session of type s is denoted $C^s \subseteq C$

For each object $c_i \in C$, let the set of objects whose instances may exchange messages with an instance of c_i be denoted as

$$C_i = \{c_j \in C \mid (M_i^{client} \cap M_j^{server}) \cup (M_j^{client} \cap M_i^{server}) \neq \varnothing\}$$

We also denote the set of all possible pairings of objects whose instances may exchange messages during a service session of type s as

$$P^s = \{(c_i,c_j) \in C \mid c_i \in C^s \text{ and } c_j \in C_i^s\}$$

where C_i^s is equal to the set $C_i \cap C^s$

Finally, for each pair of computational objects $(c_i,c_j) \in P^s$, let the set of all messages that are exchanged between instances of c_i and c_j, during execution of a service session of type s, be denoted as

$${}^sM_{ij} = (M_i^{client} \cap M_j^{server}) \cup (M_j^{client} \cap M_i^{server})$$

5 1 2 2 Processing Costs

Each message $k \in M$ that is passed from processor $n \in N$ to processor $m \in N$ has the following associated execution workloads

$\tau_n^{encode}(k)$ execution time for protocol encoding of message k on processor n

$\tau_n^{client}(k)$ execution time for client-side processing of message k on processor n

$\tau_m^{decode}(k)$ execution time for protocol decoding of message k on processor m

$\tau_m^{server}(k)$ execution time for server-side processing of message k on processor m

The total processing time on processor n due to all communication between an instance of computational object c_i on processor n and an instance of computational object c_j on processor m during one service session of type s is denoted as

$$ {}^{s}_{m}\tau^{n}_{ij} = \sum_{\forall k \in {}^{s}M_{ij}^{c_i(client)}} \{\tau_n^{encode}(k) + \tau_n^{client}(k)\} + \sum_{\forall k \in {}^{s}M_{ij}^{c_i(server)}} \{\tau_n^{decode}(k) + \tau_n^{server}(k)\} $$

$$ where \quad {}^{s}M_{ij}^{c_i(client)} = {}^{s}M_{ij} \cap M_i^{client} \quad and \quad {}^{s}M_{ij}^{c_i(server)} = {}^{s}M_{ij} \cap M_i^{server} $$

Note that, in the above, encoding and decoding times, when computational object instances are executing on the same processor, may or may not be zero depending on the application and communication protocol implementation details The total processing time on processor m for the same pair of object instances is similarly defined and denoted ${}^{s}_{n}\tau^{m}_{ji}$ We denote the sum of these times as

$$ {}^{s}\tau_{ij}^{nm} = {}^{s}_{m}\tau^{n}_{ij} + {}^{s}_{n}\tau^{m}_{ji} $$

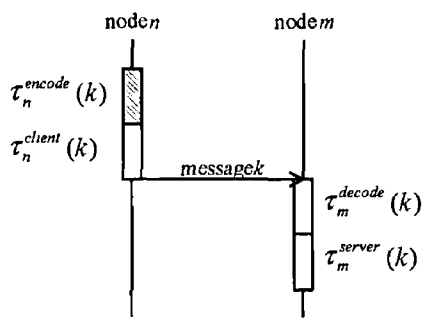


Figure 5 2 a Message Processing Times

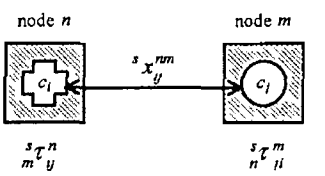


Figure 5 2 b Workflow Notation

5 1 2 3 Workflows

We define ${}^{s}w_{ij}^{nm}$ as the total processing time in the system, per unit volume of service requests of type s offered to the network, due to all messages passed between the instance of object $c_i \in C^s$ on processor n and the instance of object $c_j \in C^s$ on processor m during service sessions of type s We denote the associated bi-directional work flow as

$$ {}^{s}x_{ij}^{nm} = \frac{{}^{s}w_{ij}^{nm}}{{}^{s}\tau_{ij}^{nm}} \quad \text{(type } s \text{ sessions per second)} $$

5 1 3. Users and Service Requests

All service requests to the system originate from instances of *service initiating* objects, the set of which is denoted $C_{init} \subset C$ Each service type $s \in S$ has exactly one associated initiating object $c^s_{init} \in C_{init}$ This object may be considered as representing all users of service type s where an instance of the object on a processor represents one or more such users connected to that processor Thus, workflows from c^s_{init} represent the total of all traffic from the set of users it represents All processing costs for the processor, that are associated with users requesting and interacting with other objects during service sessions, must be accounted for in the relevant processing times defined in §5 1 2 2

Given that all the flows involving c^s_{init} are included in the formulation of ${}^s x^{nm}_{ij}$, the total volume of type s service requests to the system, from an instance of $c_i \in (C_{init} \cap C^s)$ on processor $n \in N$ is denoted as

$$\Lambda^n_s = \sum_{\substack{\forall m \in N_j \\ \forall c_j \in C^s_i}} \frac{{}^s x^{nm}_{ij}}{d} \qquad where\ d = \sum_{\forall c_j \in C^s_i} 1$$

Table 5 1 Summary of Optimisation Model Variables

S	set of all service types
C	set of all computational objects in network
C^s	set of all objects required to execute a service session of type s is denoted
C_i	set of objects whose instances may exchange messages with an instance of object c_I
C^s_i	set $C_i \cap C^s$
C_{init}	set of service initiating computational objects
P^s	set of all possible pairings of objects whose instances may exchange messages during a service session of type s
N	set of all processors in network
N_{user}	set of processors that supports direct connections from system users
N_i	set of processors that support the computational object $c_i \in C$
${}^s_n \tau^m_{ji}$	total processing time on processor n due to all communication between an instance of computational object c_i on processor n and an instance of computational object c_j on processor m during one service session of type s
${}^s x^{nm}_{ij}$	bi-directional work flow – type s sessions per second - main optimisation variable
Λ^n_s	the total volume of type s service requests to the system

5 1 4 Optimising Object Placement

The problem of optimally assigning object instances to system processors may be summarised as follows Given a system consisting of a set of connected processors N, a set of

computational objects C and a set of service graphs M specifying interactions between objects for each service, find the set of flows $\{{}^{s}x_{ij}^{nm}\}$ that maximises the total service request volume to the system under the following constraints

- fractional volumes of accepted requests from each user are a given value (C1)
- work flows between all object instances are conserved (C2)
- each processor's utilisation must not exceed a given limit (C3)

The optimal allocation of objects to processors may be easily determined by examining the set of solution flows for this problem The problem formulation initially assumes that any object instance may exist on any processor, with the exception of instances of the initiating objects whose location is fixed according to user connections If, in the solution, all flows associated with an object instance are zero, then that object may be eliminated from the processor

The objective of maximising the service request volume ensures that the maximum system throughput is obtained The set of constraints (C1) may be expressed as

$$\frac{\Lambda_s^n}{\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \Lambda_s^n} = \alpha_s^n \qquad \forall n \in N_{user}, s \in S \qquad where \sum_{\substack{\forall n \\ \forall s}} \alpha_s^n = 1 \qquad \text{(C1)}$$

If it is assumed that the location of object instances is a permanent design choice, it is necessary to specify the service mix $\{\alpha_s^n\}$ in accordance with careful consideration of the expected, long-term user demands as object allocation can have a large impact on the achievable performance of the system

The set of constraints (C2) are necessary to relate the set of flows $\{{}^{s}x_{ij}^{nm}\}$ between instances of objects and between users of the system Firstly, we define the set of all objects that are common to object pairs

$$C^{s}_{common} = \{c_j \in C^s \mid (c_i,c_j),(c_{i'},c_j) \in P^s \quad i,i' \neq j\}$$

The constraint equations to enforce balancing of all flows in the system may now be written by taking each object instance c_j in the system and equating the total flows between this object instance and all instances of two other objects c_i and $c_{i'}$

$$\sum_{\forall n \in N_i} {}^{s}x_{ij}^{nm} = \sum_{\forall n \in N_{i'}} {}^{s}x_{i'j}^{nm} \qquad \forall s \in S,\ \forall c_j \in C^s_{common},\ \forall (c_i,c_{i'}) \in C_j,\ \forall m \in N \qquad \text{(C2)}$$

The set of constraints (C3) is determined by limiting the total processing time at each processor for all object instances residing on the processor

$$\sum_{\forall s\in S}\sum_{\forall c_i\in C^s}\sum_{\forall c_j\in C_i^s}\sum_{\forall m\in N_j} {}^{s}_{m}\tau_{ij}^{n}\ {}^{s}x_{ij}^{nm} \le \rho_{\max}^{n} \qquad \forall n\in N \qquad \text{(C3)}$$

where the $\rho_{\max}^{n}$ is the maximum allowable utilisation of processor n As the objective function and all constraints are linear, the problem may be formulated as the following LP problem

$$\begin{aligned} &\textit{Maximise} && \sum_{\substack{\forall n\in N_{user}\\ \forall s\in S}} \Lambda_s^n \\ &\textit{Subject to} && (C1),(C2),(C3),\forall\ {}^{s}x_{ij}^{nm} \ge 0 \end{aligned} \qquad \text{(M1)}$$

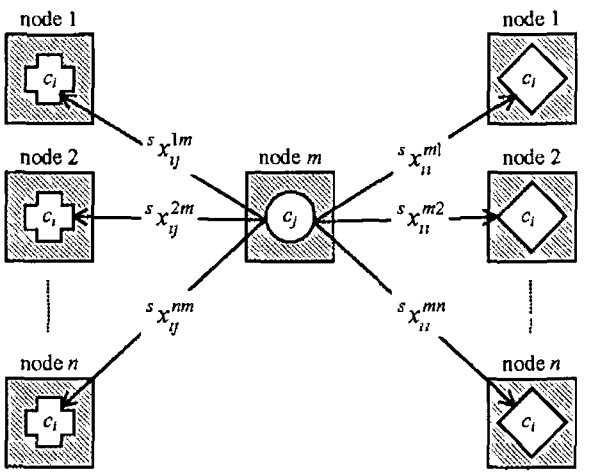


Figure 5 3 Constraint (C2) Workflow Balance Requirement

Flows on left must balance with flows on right Flow balance is specified by relating flows between a component and two other components with which it communicates

5 1 5. Optimising Object Placement with Installation Costs

Generally, the solution to the previous problem yields a large number of object instances distributed across the processors This is mainly due to object instances tending to cluster on processors to avoid the added costs associated with inter-processor communication Also, in order to achieve the maximum service request volume, the largest possible amount of processing capacity in the system tends to be used by distributing these clusters to as many processors as possible This situation may be limiting in practice if the costs of installation, licensing and maintenance of software components are high Also, the solution can give instances of objects, which have very low utilisation and subsequently the cost of installation may outweigh their benefit to the system To address these issues, an installation cost is associated with each object instance in the system A constraint may then be formulated to limit the total installation cost for the network, which has the effect of limiting the number of object instances in the system

The presence or absence of object instances on processors is represented as a set of 0-1 integer variables (decision variables) and is defined by the set

$$Y=\{y_i^n \in \{0,1\} \mid \forall c_i \in C, \forall n \in N_i\}$$

We define the set of related installation costs as

$$\{e_i^n \in \Re \mid \forall c_i \in C, \forall n \in N_i\}$$

The total flow between object instance c_i on processor n and all other object instances is calculated as

$$\bar{x}_i^n = \sum_{\forall s \in S} \sum_{\forall c_j \in C_i} \sum_{\forall m \in N_j} {}^s x_{ij}^{nm}$$

Each binary variable is related to the corresponding total flow, giving the set of constraints (C4) below Note that the threshold value $\bar{x}_{threshold}$ allows object instances with very low utilisation to be eliminated from the system

$$y_i^n = \begin{cases} 0 & \bar{x}_i^n < \bar{x}_{threshold} \\ 1 & \bar{x}_i^n \geq \bar{x}_{threshold} \end{cases} \qquad \forall y_i^n \in Y \qquad \text{(C4)}$$

Limiting the total installation costs to a desired value E, gives the constraint

$$\sum_{\forall c_i \in C} \sum_{\forall n \in N_i} e_i^n y_i^n \leq E \qquad \text{(C5)}$$

Note that there is a lower bound on E beyond which there is no feasible solution Adding constraints (C4) and (C5) to the original problem gives the MIP problem

$$\begin{array}{ll} \underset{{}^s x_{ij}^{nm}}{Maximise} & \sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \Lambda_s^n \\ Subject\ to & (C1),(C2),(C3),(C4),(C5), \forall\ {}^s x_{ij}^{nm} \geq 0 \end{array} \qquad \text{(M2)}$$

5 1 6 Optimising Random Splitting and Admission Control

The solution obtained from problem (M1) or (M2) gives the optimal distribution of flows between object instances for the given relative input traffic levels specified by (C1) In order to maintain the system at the optimum, it is necessary to (a) limit the service request volumes by rejecting a portion of new service requests so that the solution input traffic levels are maintained and (b) ensure that the optimal flows between object instances are maintained Condition (a) is restrictive and may lead to large proportions of service requests being

rejected when the relative input traffic levels vary from (C1) over time This restriction may be alleviated by dynamically re-optimising the system periodically This generates a new problem (M3) where the object instance locations are fixed by the solution of either (M1) or (M2) and the flows are re-optimised given an estimate of the current offered input traffic intensities The solution also generates the re-optimised maximum input traffic levels The (M3) problem is a variation of (M1) where all sets of object instance locations N_i are fixed by the solution of (M1) or (M2) and constraint set (C1) is modified as follows

$$\frac{\Lambda_s^n}{\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \Lambda_s^n} = \frac{\overline{\lambda}_s^n(k)}{\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \overline{\lambda}_s^n(k)} \qquad \forall n \in N_{user}, s \in S \qquad \text{(C1)'}$$

where $\{\overline{\lambda}_s^n(k)\}$ is the set of estimates of the offered traffic intensities from users expected over the next T seconds This may be estimated simply as $\{\overline{\lambda}_s^n(k)\} = \{\lambda_s^n(k-1)\}$ where $\{\lambda_s^n(k-1)\}$ is the set of actual offered traffic intensities measured over the previous T seconds The optimisation (M3) is run at the start of each control period kT, having received $\{\lambda_s^n(k-1)\}$ from initiating objects, and the resulting solution is distributed to processors as follows

(a) Each initiating object $c_i \in (C_{init} \cap C^s)$ receives the set of acceptance probabilities $\{_{acpt}p_s^n(k)\}$ for new service requests of type s over the next T seconds where

$$_{acpt}p_s^n(k) = \begin{cases} \dfrac{_{sol}\Lambda_s^n(k)}{\overline{\lambda}_s^n(k)} & _{sol}\Lambda_s^n(k) < \overline{\lambda}_s^n(k) \\ 1 & _{sol}\Lambda_s^n(k) \geq \overline{\lambda}_s^n(k) \end{cases}$$

and where $\{_{sol}\Lambda_s^n(k)\}$ is the solution set of $\{\Lambda_s^n\}$ at time kT These probabilities are used to implement Percentage Thinning (PT) of arrivals over the next T seconds

(b) Each object instance receives the set of optimal flows $_{sol}{}^s x_{ij}^{nm}(k)$ (the solution value of ${}^s x_{ij}^{nm}$ at time kT) between itself and all other object instances with which it may communicate The optimal flow solution is implemented by an object instance as follows When an object instance c_i on processor n is required to send a service message to an instance of an object c_j, in order to continue a service session of type s, it chooses a processor according to Bernoulli trials where the probability of choosing c_j on processor m is determined by the *random splitting* probability

$$_{split}{}^s p_{ij}^{nm}(k) = \frac{_{sol}{}^s x_{ij}^{nm}(k)}{\sum_{\forall m \in N_j} {}_{sol}{}^s x_{ij}^{nm}(k)}$$

This choice applies only to the first time during a service session that object c_j is required All subsequent messages requiring c_j during the same service session are sent to the same object instance This scheme will maintain average flows at the correct values

5.1 7. Optimising Network Revenue with Fairness

In a multi-service network it is often desirable that service requests should receive priority in relation to the revenue generating ability of the service types However, implementing revenue weightings can lead to unfair treatment of customers when the offered arrival volumes for each service are disproportionate to their respective weightings These issues are taken into account with a modification to the cost function to allow revenue weighting and a modification to constraint set (C1)' to allow a degree of fairness to be specified

Normally, in revenue maximisation problems, the *probability of acceptance* of service requests is the optimisation variable (such as in [Lodge, 1999] for Intelligent Network load control) and the maximisation takes the following form

$$\underset{P^s_{accept}}{Maximise} \qquad \sum_{\forall s} r_s P^s_{accept} \overline{\lambda}_s(k)$$

Here, r_s is the revenue weighting associated with completing a service session of type s, $\overline{\lambda}_s(k)$ are the expected traffic intensities for period k for service s and P^s_{accept} is the probability of a service session of type s being accepted The objective is thus to maximise the expected total revenue for the network A similar objective is employed here but the probability of acceptance is expressed in terms of the flow variable ${}^s x^{nm}_{ij}$ of our original problem The probability of acceptance in the flow variable is

$$_{acpt}P^n_s(k) = \frac{\Lambda^n_s}{\overline{\lambda}^n_s(k)} \qquad 0 \le \frac{\Lambda^n_s}{\overline{\lambda}^n_s(k)} \le 1$$

and the new objective function for the problem may be stated as

$$\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} r_s \frac{\Lambda^n_s}{\overline{\lambda}^n_s(k)} \overline{\lambda}^n_s(k) \qquad s.t \qquad 0 \le \frac{\Lambda^n_s}{\overline{\lambda}^n_s(k)} \le 1$$

This objective and constraint may be rewritten as

$$\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} r_s \Lambda^n_s \qquad s.t \qquad \Lambda^n_s \le \overline{\lambda}^n_s(k), \forall\ {}^s x^{nm}_{ij} \ge 0 \qquad \text{(C6)}$$

where $\{r_s \in \Re \mid s \in S\}$ are the revenue weightings associated with completed service sessions and $\overline{\lambda}_s^n(k)$ is the expected traffic intensity for the next T seconds As the relative traffic volumes can no longer be constant, constraint set (C1)' is modified to

$$\delta_f \leq \frac{\overline{\lambda}_s^n(k)}{\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \overline{\lambda}_s^n(k)} \Bigg/ \frac{\Lambda_s^n}{\sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} \Lambda_s^n} \leq \frac{1}{\delta_f} \quad \forall n \in N_{user}, s \in S \qquad where\ 0 < \delta_f \leq 1$$

(C1)''

where δ_f controls the amount by which the relative traffic volumes may vary from the expected relative volumes of offered traffic Values of δ_f close to 1 ensure a fair treatment of customers whilst values close to 0 potentially allow unfair treatment but higher total revenue for the system The objective and all constraints are linear thus giving an LP problem

$$\begin{aligned} &\textit{Maximise} && \sum_{\substack{\forall n \in N_{user} \\ \forall s \in S}} r_s \Lambda_s^n \\ &\textit{subject to} && (C1)'', (C2), (C3), (C6) \forall\ {}^s x_{ij}^{nm} \geq 0 \end{aligned} \qquad \text{(M4)}$$

Note that, again, all sets of object instance locations N_i are fixed by the solution of (M1) or (M2) The solution distribution and implementation strategy is as described in the previous section Note also that the revenue optimisation constraint and objective function may be applied to the formulation for optimisation of object placement in problems (M1) and (M2) However, such a modification is not considered here as it is assumed that the network is dimensioned so that it operates in the under-loaded region a large proportion of the time and revenue optimisation is advantageous only when offered traffic would cause the desired network load to be exceeded It could be employed to the allocation problem if the normal operating point of the network is at offered loads higher than the throttling levels

5 1 7 1 Adjustment to Revenue Optimisation Algorithm

Unlike in the throughput maximisation problems (M1, M2 and M3), the revenue maximisation algorithm constrains the maximum arrival rates to be less than or equal to the expected arrivals When arrival rates are low compared to the maximum network throughput, the optimisation problem (M4) will not tend to drive all nodes to full capacity and the resulting random splitting may give uneven loading across processors Our view is that load balance is a desirable property once it does not artificially constrain the maximum network throughput (This has been the case with the throughput maximisation problems) We assert that it is not possible to assure load balance, by addition of further linear constraints to the above problem, without reducing the possible total system revenue Thus, we make an

adjustment to the optimisation algorithm, in the form of a simple two phase heuristic, to restore the tendency for load balance in the solution

PHASE 1

solution = solve (M3)

if $_{sol}\Lambda_s^n(k) = \overline{\lambda}_s^n(k) \quad \forall n \in N_{user}, s \in S$ then goto END

PHASE 2

solution = solve (M4)

END

The non-revenue problem (M3) is solved If all arrivals are accepted in the solution, then the solution is revenue optimal and fair, regardless of revenue or fairness weightings, and there is no need to solve (M4) If not, then (M4) is solved and gives the final revenue-optimal solution Only solving (M4) when throttling is required will tend to drive the solution towards load balance Note that, assuming that the network operates at less than capacity for the majority of the time, PHASE 2 will be called relatively infrequently and on average the algorithm complexity does not increase substantially

5.2. Co-operative Market-Based Algorithm for Load Control

In the previous section a method of optimising object placement and load control was introduced This section describes an alternative sub-optimal market-based strategy for distributing load and load limiting in a network of optimally placed computational objects This method was originally used for control of Intelligent Networks [Jennings, 2001] but has been adapted and extended here to control in distributed object scenarios

5 2 1 Strategy Overview

Load sharing and admission control in this method is effected by means of *tokens* A token type is associated with each pair of communicating computational object instances If, during the course of a service session, an object instance requires communication with another object in order to continue the session, it must posses a token of the relevant type to do so See Figure 5 4 below This token is then considered 'spent' and is removed from a pool of available tokens Tokens are consumed on a per-session basis That is, once an object has 'spent' a token to allow communication with another object, it may continue message passing with this object to complete that service session but may not reuse it during any future session

Thus, each new service-initiating request from a user requires a certain set of tokens within the network in order to complete the service session This set of tokens is referred to as a

token chain Note that the first token in a chain (referred to as the *initiating token*) will apply to acceptance or rejection of the initial service request from the user That is, if a token of this type is not available, the request is immediately rejected Otherwise the service session is accepted As will be explained below, tokens are allocated in such a manner that, once a session is accepted, sufficient tokens exist to complete it

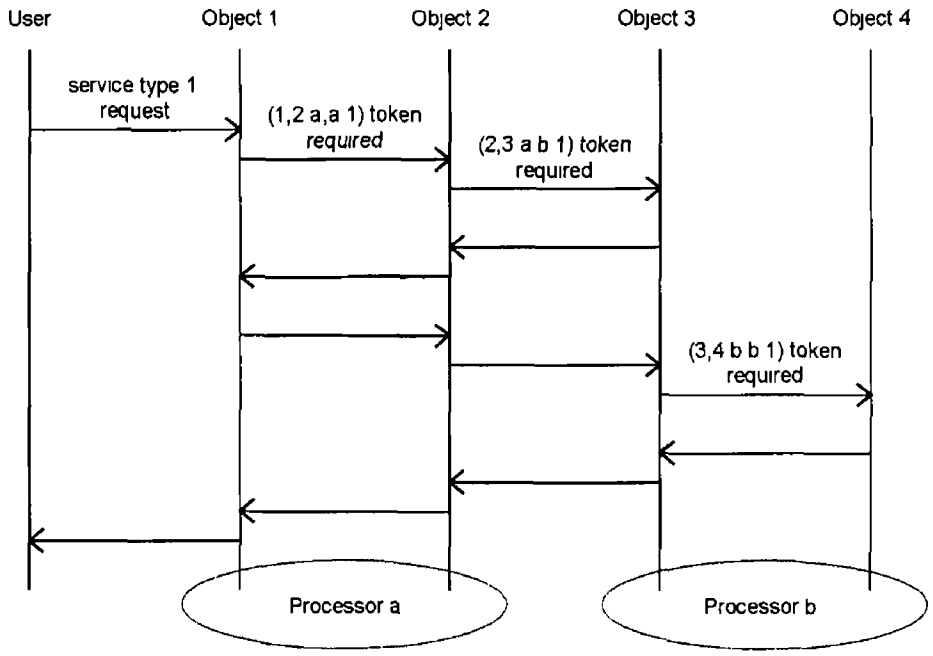


Figure 5 4 Example of Token Use During a Service Session

A token is required to allow sending of the first forward message in a session from one object to another A token is not required to receive this message nor to send or receive any subsequent messages in the session

In order to co-ordinate token usage, each processor in the network has an associated *load control agent* that maintains a pool of tokens of various types on behalf of the computational objects that the processor executes The 'spending' of a token has associated with it a processing cost on the processors that host the associated objects Thus, by limiting the collection of tokens associated with a processor during a certain control period, it is possible to limit mean processor utilisation Also, the distribution of tokens across the network processors can be used to control the distribution of load and the admission of new requests into the network

In order to maintain a desired number of tokens in the token pools, a central *auction* is run every T time units At the beginning of this auction process, load control agents submit *bids*, which consist of the average arrival rate of new service session requests, for each service type, expected over the next control period It is assumed that the auction process has knowledge of the average available processing capacity on each processor and the processing

requirements associated with each possible token type The auction process then executes with this information as input parameters and allocates sets of tokens to each processor's load control agent for use during the next control period

During the auction process, tokens are allocated in chains That is, an initiating token is chosen first and then all remaining tokens required to complete a session of this service class are chosen The choice of tokens is governed by a market-based auction algorithm, which allocates the most profitable tokens available each time The process continues until all possible processing capacity is used for the next T seconds Pools of tokens are then distributed to the load control agents for spending over the next T seconds Note that it is assumed that the auction process completes in a time much less than T, so that processors do not wait any significant amount of time for token pools to be refilled Note also that when a pool is refilled, any existing tokens from the previous control period are removed These bidding and auctioning processes are described more formally below

5 2 2 Notation

In order to describe the market-based algorithm in detail the following notation is defined

- There are K types of computational objects in the network Let k denote an arbitrary object
- There are I load control agents in the system, each associated with one processor Let i denote an arbitrary agent
- There are J different service classes Let j denote an arbitrary service class
- Each service class j has associated with it a profit value $r(j)$, i e each successfully completed service session of type j earns $r(j)$ profit units for the network
- Tokens in the system are denoted by the tuple (k,k',i,i',j) This denotes a token that permits an object of type k, residing on node i, to communicate with an object of type k', residing on node i', for the purpose of completing one service session of service class j Node i is designated as the *initiating* node and node i' as the *target* node for this token
- During the auction, a tally is kept of the amount of capacity that remains unused on each processor Let m_i denote the remaining processing capacity on processor i at any stage during the auction process
- A record is kept of the satisfied demands (in terms of tokens granted) for each processor Let $n(k,k',i,i',j)$ denote the number of (k,k',i,i',j) type tokens allocated, at any instance during the auction process
- A record is also kept of the number of class j token chains allocated that have their initiating token on processor i Let $n(i,j)$ denote the number of such tokens which have been allocated, at any stage during the auction process

- The set C_{ij} is defined as the set of all allocated class j tokens in all chains which originated from initiating tokens on processor i
- The quantity $p_i(k,k',i,i',j)$ is defined as the processing cost incurred on node i due to a (k,k',i,i',j) token being spent Similarly, the quantity $p_{i'}(k,k',i,i',j)$ is defined as the processing cost incurred on node i' due to a (k,k',i,i',j) token being spent These processing costs may be derived from message processing and protocol encoding/decoding times, similar to the methods given in §5 1 2 2 for estimating the quantities ${}^{s}_{m}\tau^{n}_{ij}$ and ${}^{s}_{n}\tau^{m}_{ji}$
- The quantity $\bar{\lambda}(i,j)$ is defined as the expected rate of arrivals to node i for service initiation requests of service type j, over the next T time units

5 2 3 Load Control Agent Bids

Load control agents submit bids to the auction agent every T time units in order to receive tokens for use over the next control period The bid for agent i consists of its available processing capacity over the coming period of T time units, denoted c_i, and the rate of new service requests expected from users to this processor, $\bar{\lambda}(i,j)$ The available processing capacity may be given any desired value in order to maintain loading at or below a desired value The expected rate of new service requests may be estimated by the relevant agents by simply taking the average measured rate over the previous control period

5 2 4. The Auction

In order to choose the most profitable tokens to assign, the auction calculates an expected *marginal utility* of each additional token that may be allocated during an auction The expected marginal utility is the ratio of the expected *marginal gain* to the expected *marginal cost* of a token These are defined below

5 2 4 1 Expected Marginal Cost for Initiating Tokens

The expected *marginal cost* associated with allocating an additional *initiating* token of type (k,k',i,i',j) is defined as the estimated processing cost, relative to the remaining processing capacities on the relevant processors, that would be spent in the network when executing the entire token chain resulting from this token being used to accept a new service session The total processing cost is estimated by taking the average total cost over all previously allocated chains which originated from tokens of type (k,k',i,i',j) This cost is calculated as

$$v(k,k',i,i',j) = \sum_{\forall(\bar{k}\ \bar{k}'\ \bar{i}\ \bar{i}'\ \bar{j})\in C_{\bar{i}\bar{j}}} \left(\frac{p_{\bar{i}}(\bar{k},\bar{k}',\bar{i},\bar{i}',\bar{j})}{m_{\bar{i}}} + \frac{p_{\bar{i}'}(\bar{k},\bar{k}',\bar{i},\bar{i}',\bar{j})}{m_{\bar{i}'}} \right) \Big/ n(\bar{i},\bar{j})$$

If no token chains originating from token type (k,k',i,i',j) have previously been allocated, then this quantity is calculated from the costs associated with the initiating token only That is

$$v(k,k',i,i',j)=\frac{p_i(k,k',i,i',j)}{m_i}+\frac{p_i(k,k',i,i',j)}{m_i}$$

5 2 4 2 Expected Marginal Cost for Non-Initiating Tokens

The expected *marginal cost* associated with allocating an additional non-initiating token of type (k,k',i,i',j) is defined as the total processing cost expended by the network, if the token is consumed, relative to the remaining processing capacities on the relevant processors This marginal cost is calculated as follows

$$v(k,k',i,i',j)=\frac{p_i(k,k',i,i',j)}{m_i}+\frac{p_i(k,k',i,i',j)}{m_i}$$

5 2 4 3 Expected Marginal Gain for Initiating Tokens

The expected *marginal gain* associated with allocating an additional *initiating* token of type (k,k',i,i',j), given the amounts of tokens of this type that have already been allocated during an auction, is defined as the profit associated with consuming it times the probability that it will be consumed over the next control interval This probability is equivalent to the probability that there that there will be at least $n(i,j)+1$ class j arrivals over the next T time units at processor i If we assume that the arrival process of new service initiating requests to a processor is a Poisson process, we may calculate this probability as follows

Given that the probability of a class j arrivals in T time units for a Poisson process will be

$$P_a(T)=\frac{(\bar{\lambda}(i,j)T)^a}{a!}e^{-\bar{\lambda}(i,j)T},$$

then the probability that there will be at least $n(i,j)+1$ class j arrivals over T time units is calculated as

$$\mathrm{P}[a\geq n(i,j)+1]=\sum_{a=n(i,j)+1}^{\infty}\frac{(\bar{\lambda}(i,j)T)^a}{a!}e^{-\bar{\lambda}(i,j)T}$$

The expected marginal gain of an initiating token may now be defined as

$$u(k,k',i,i',j)=r(j)\sum_{a=n(i,j)+1}^{\infty}\frac{(\bar{\lambda}(i,j)T)^a}{a!}e^{-\bar{\lambda}(i,j)T}$$

This calculation of marginal gain is identical to that described in [Jennings, 1999]

5.2.4.4 Expected Marginal Gain for Non-Initiating Tokens

Given that the initiating token for a chain is assigned before any other tokens in the chain, the probability of consuming any other token in the chain is 1. In this case, the expected *marginal gain* associated with allocating an additional non-initiating token may be any constant value. A constant value of 1 is chosen.

5.2.4.5 Expected Marginal Utilities

The expected *marginal utility* of allocating an additional token of type (k,k',i,i',j) may now be defined as the marginal gain per marginal cost of such an allocation and is defined as

$$\delta(k,k',i,i',j) = \frac{u(k,k',i,i',j)}{v(k,k',i,i',j)}$$

where $\delta(k,k',i,i',j)$ expresses the derivative of the utility function with respect to the relative processing required for a (k,k',i,i',j) token allocation. The auction algorithm aims to maximise total overall utility by distributing the resources in a series of allocations such that each allocation results in a maximal increase in overall utility. The best allocation in each step is thus the one with the highest derivative. The auction algorithm is described in detail in the following section.

5.2.4.6 The Auction Algorithm

1) Initialisation

Reset token allocations $n(k,k',i,i',j) = 0$ for all k,k',i,i',j

Reset initiating token counts $n(i,j) = 0$ for all i,j

Set processing capacities remaining $m_i = c_i$ for all i

Set marginal gains for initiating tokens

with $n(i,j) = 0$, $u(k,k',i,i',j)\big|_{n(i,j)=0} = r(j)\left(1 - e^{-\bar{\lambda}(i,j)T}\right)$ [1]

Set marginal gains for non-initiating tokens $u(k,k',i,i',j) = 1$

Set marginal costs $v(k,k',i,i',j)$ for all tokens

---

[1] Noting that

$$\begin{aligned} u(k,k',i,i',j)\big|_{n(i,j)=0} &= r(j)\sum_{a=1}^{\infty}\frac{(\bar{\lambda}(i,j)T)^a}{a!}e^{-\bar{\lambda}(i,j)T} \\ &= r(j)\left[\sum_{a=0}^{\infty}\frac{(\bar{\lambda}(i,j)T)^a}{a!}e^{-\bar{\lambda}(i,j)T} - e^{-\bar{\lambda}(i,j)T}\right] \\ &= r(j)[1 - e^{-\bar{\lambda}(i,j)T}] \end{aligned}$$

2) Allocate an Initiating Token

Find all initiating tokens that maximise $\delta(k,k',i,i',j) = u(k,k',i,i',j)/v(k,k',i,i',j)$

Choose token $(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$ from these at random

Update token allocations $n(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j}) = n(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j}) + 1$

Update initiating token count $n(\hat{i},\hat{j}) = n(\hat{i},\hat{j}) + 1$

Recalculate marginal gains $u(k,k',\hat{i},i',\hat{j})$ of all initiating tokens for updated $n(\hat{i},\hat{j})$

Update remaining capacities $m_i = m_i - p_i(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$ and $m_i = m_i - p_i(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$

Recalculate marginal utility for chosen token type

3) Allocate Remaining Tokens in Token Chain

While there are remaining tokens to allocate in this token chain

Find all token types that maximise $\delta(k,k',i,i',j) = u(k,k',i,i',j)/v(k,k',i,i',j)$

Choose token type $(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$ from these at random

Update token allocations $n(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j}) = n(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j}) + 1$

Update remaining capacities $m_i = m_i - p_i(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$ and

$m_i = m_i - p_i(\hat{k},\hat{k}',\hat{i},\hat{i}',\hat{j})$

Recalculate marginal utility for chosen token type

end while loop

4) Loop

do while not 0 tokens allocated in last pass and all m_i not 0

goto step 1)

5) Distribute Tokens

Distribute allocated tokens of type (k,k',i,i',j) to load control agent on node i

5 2.5. Token Spending for Initiating Tokens

Initiating tokens control acceptance or rejection of new service control sessions entering the network over the course of a control period Under heavy load there are not an adequate number of tokens available to accept all service sessions during the control period Under these conditions, tokens will tend to exhaust early during a control period causing undesirable traffic patterns (burstiness) within the network To avoid this, a rationing strategy is required to spread the available tokens more evenly over the control period The rationing process used

here is adopted from Jennings [2001] and employs Percentage Thinning (PT) to regulate the acceptance of service requests At sub-intervals of the control interval PT coefficients for each service type are updated, using estimates of the number of requests that will arrive before the end of the control interval and the number of remaining tokens Arriving service requests are subjected to a PT throttle using the relevant PT coefficient as parameter This parameter is calculated as follows

Let τ denote the length of the sub-intervals, τ is chosen such that $T = A\tau$, where A is some integer Let $a \in (1, A)$ denote the current sub-interval number Let $\gamma^*(j)$ denote the estimated number of arrivals of requests for service type j until the end of the control interval and let $n'(j)$ denote the number of tokens remaining for service type j Let $m_j(a)$ denote the number of requests for service type j that arrived during sub-interval a and let $m'_j(a)$ denote the number of these that were accepted Finally let $p^a(j)$ denote the probability of acceptance of a request for service type j (the PT coefficient) The algorithm contains two steps the first (Initialisation) is executed at the start of the control interval and the second (Update PT coefficients) is executed at the start of each sub-interval

1) Initialisation

Set $a = 1$

For all service types $j = 1, \ldots, J$ do

Set $\gamma^*(j) = q_k(j)$, where $q_k(j)$ is the number of requests for service type j that arrived over the duration of the previous control interval

Set $n'(j) = n_k(j)$, $m'(j) = 0$

Set $p^a(j) = \min(1, n'(j)/\gamma^*(j))$

2) Update PT coefficients

For all service types $j = 1, \ldots, J$ do

Set $n'(j) = n'(j) - m'_j(a)$

Set $\gamma^*(j) = \frac{(A-a)}{a}\sum_{a=1}^{a} m_j(a')$

Set $p^a(j) = \min(1, n'(j)/\gamma^*(j))$

Set $a = a + 1$

Having calculated the PT coefficient, acceptance or rejection of a new service session request is decided as follows

Select a random number X uniformly distributed in the range (0 0, 1 0)

if $X < p_a(j)$ then Service request is accepted, else Service request is throttled

5.2.6 Token Spending for Non-Initiating Tokens

If, during the course of a service session, an object instance requires communication with another object in order to continue the session, it must posses a token of the relevant type to do so This token is then considered 'spent' and is removed from the pool of available tokens for that node There may be more than one token type to choose from i e there may be tokens in the pool for a target object available on multiple different nodes In this case a token is chosen at random in proportion to the number of *initial* allocations Note that there is no need to ration non-initiating tokens over the duration of the control period, as there will always be adequate tokens to complete a service session This is ensured by the auction process

5.3. Chapter Summary

This chapter has presented general approaches to optimal object allocation and performance controls suitable for application to the service platform model described in Chapter 4 The object placement solution described can be applied to obtain optimal placements of COs on service platform nodes This is done in Chapter 6 and the resulting placements are analysed The optimal random splitting and admission controls developed can be employed as internal and external performance controls This approach is investigated in Chapter 6 Also investigated in the next chapter, is the performance of the sub-optimal market algorithm compared to the optimal controls

Chapter 6. Analysis of Service Platform and Performance Controls

In this chapter, the performance properties of the algorithms proposed in Chapter 5 are examined The properties of the Computational Object allocation method are examined and a distribution is chosen to complete the service platform definition An analytic model is developed and the accuracy of the simulator verified The performance of the optimal and market internal and external performance controls is examined and compared to the results for a simple load-balancing scheme

6.1. Optimal Allocation of Computational Objects

An allocation of COs to processing nodes is required to complete the definition of the service platform model for simulations and analysis Here we describe the implementation and examine the properties of the optimal static CO allocation strategy developed in §5 1 and then choose an allocation for examination of the dynamic algorithms (optimal and market-based random splitting and admission control) We have assumed that, once assigned, the CO allocation is fixed and do not consider mobility of the COs at runtime However, splitting between the allocated CO copies and the Percent Thinning coefficient values of the gateway throttles are variable and are determined by the dynamic algorithms (Optimal and Market) These are discussed in the remainder of this chapter We first describe the implementation of the CO allocation problems This implementation also relates in part to the optimal internal control

6.1.1. Implementation of the CO Placement LP/MIP

The Linear Programs of §5 1 generate quite large problem spaces that would be difficult to construct 'by hand' Thus, *C* language code has been written for the purpose of this thesis, which takes as input the specification for the service platform and the CO interaction details It constructs the linear programming problem matrices programmatically and formats the problem for input to IBM's OSL solver (see §3 3 6) The OSL software package provides a

runtime library that is linked with our LP/MIP matrix construction code so that one executable may be obtained When the solution vector is returned by OSL, some post processing is done to extract the pertinent information and the solution is presented as simple text output We give a description of the functioning of our code here Firstly, the inputs to the program are specified as follows

Nodes The number of nodes and their operating speeds As messages are specified with respect to their nominal execution times rather than number of instructions, processor speeds are specified in terms of relative processing speeds e g a processor of speed 2 executes a message in half its nominal execution time

Services The number of service types and a textual description of each

Computational Objects A list of COs This is a numbered list with a textual description of each CO, which is used on output to aid legibility of the solution

CO Installation Costs An optional input that specifies the relative cost of installing a CO on a given node

Minimum CO Utility An optional input that specifies the minimum traffic volume a CO must serve before it is allowed to be allocated in the network

CO Allocation Constraints We wish to constrain the placement of the *GW* objects as these are collocated with the SS 7 stack in the service platform and cannot be replicated arbitrarily in the network This condition may be specified on input as a list of fixed CO assignments for the problem

CO Interactions To formulate the relationships between the main workflow variables of the LP/MIP, the code requires knowledge of the COs that interact during each service This is specified as a list of CO pairs for each service For example, referring to Figure 6 1, the CO interactions for Service A would be `(SSP,GW), (GW,SCFP), (SCFP,GSEP),` etc Note that lists for different services may contain common components (e g the *GSEP*) as well as service specific ones

Work Flow Description We also require a specification for work-flows between all object instances This relates to message specification in the LP/MIP (§5 1 2 1) Each message passed between COs generates work on their processors For each CO pair, defined above, this work is specified in the input as lists of messages passed during a service session and their corresponding processing times The input is of the form

```
[Source_CO, CO_Pair_ID, CO_1_ORB_time, CO_1_processing_time,
                                CO_2_ORB_ORB_time, CO_2_processing_time]
```

Note that 'orb time' may be either encoding or decoding time depending on the message direction, which is specified by stating the source CO Beyond this, the message direction itself is not required to construct the problem as the network flows (in the optimisation problem space) are bi-directional Thus, for example, it is equivalent to specify the target CO as the message source and reverse the order of CO_1 and CO_2 processing times in the list

Note also that, in the implementation we assume that the same message, on different processors of the same speed, executes in the same time This time is then linearly scaled according to the specification of processor speeds This simplifies the implementation, however, note that it is not a requirement of the general strategy where each message on each processor may have an independent set of processing times (see §5 1 2 1)

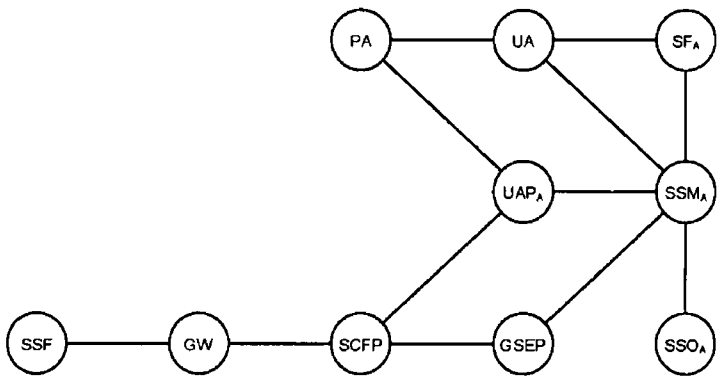


Figure 6 1 COs and Interaction Edges (Service A)

Service Users and Relative Traffic Volumes The LP/MIP is driven by maximisation of a set of input flows from users to the network (where the 'users' in our case represent groups of SSPs connected to the *GW*) A description of the service users is specified on input as a source CO and a per-service traffic value for a CO pair that contains the source CO, for example `(SSP,GW)` This is in the form

```
[source_CO, CO_pair, service_1_traffic, service_2_traffic,
            service_3_traffic]
```

Note that the source CO (the *SSP*) is automatically assumed to be a fixed-location CO That is, we assume that traffic sources are not mobile The target CO of the pair may or may not be fixed, depending on the CO allocation constraints (as describe above) The traffic volume is inputted in units of *service sessions per second* A list of such sources and their traffic volumes is specified on input

6111 Coefficient Matrix Construction

Having received the inputs, the program constructs the matrix coefficients and constraints for the LP/MIP These are as follows

Total Work Flow The workflow description from the input is processed to obtain the total 'local' and 'remote' workflow for each CO pair 'Local' workflows relate to collocated objects and exclude ORB time whilst 'remote' workflows include it In either case, this workflow is representative of the total execution time expended in the network by a pair of CO copies, during execution of one service session (see §5 1 2 3)

Input Traffic Constraints All user flows are fixed relative to each other with a set of real-variable equality constraints The input traffic specification is transformed to the workload flow variable by normalising it with respect the work flow value of the user CO pair For example, in a single service network, the input specifies 2 sources as follows 1 session per second from `SSP1` to `GW1` and 2 sessions per second from `SSP2` to `GW2` And the total work flow during a service session for `(SSP1,GW1)` is 0 5 seconds and the total workflow for `(SSP2,GW2)` is 0 2 seconds The constraint is thus `5 x (SSP1,GW1) = (SSP2,GW2)` Constraints are created to relate each source for each service type in the network Any one of these source flow variables may be chosen as the objective function for maximisation as this will maximise the total arrival rate

Interaction of Edge Pairs The program constructs the pairings of flows between COs, so that each flow in the graph can be inter-related with all others by specifying constraints (as per §5 1 2 1) This is a construction of a list of *pairs of CO pairs* for each service, for example in the form

```
[service_1 ((SSP,GW),(GW,SCFP)) ((GW,SCFP),(SCFP,GSEP))   ]
```

This list must include all edge pairs in the service interaction graph (Figure 6 1 is an example of the service graph for Service A) (Redundant pairs will make the problem space larger unnecessarily and are avoided) From this graph, all edges in the service interaction graph are inter-related by constructing one real-variable equality constraint for each pair of pairs, and for each allocation of non-common COs to each network node (see §5 1 4) These constraints are specified as per the total workflows Either local or remote workflow values are used depending on whether the particular constraint relates to objects on the same or different nodes Note that the user traffic pairs are also included in the graph, so that all flows are directly or indirectly related to the input traffic

Node Capacity Constraints These are inequality constraints, limiting the total workflows associated with a node (representing the utilisation of the node) as per §5 1 4

Installation Costs Finally the installation costs as per §5 1 5 are added to the constraints These relate real variables to integer variables If constraints are not specified in the input then the problem is constructed as an LP otherwise it is constructed as a 0-1 MIP

A general overview of the LP/MIP coefficient matrix is shown in Table 6 1 Note that this is only an estimate of the constraint dimensions, as there are some subtleties dependent on the user input For example, if fixed-location COs are specified, then they do not appear in all edge pair constraints (Their flows may instead be set to zero using column constraints, where appropriate) Note that, although this is a large matrix it is sparse and problems with several thousand variables solve in the order of seconds

Real Variables																Integer Variables											
Edge 1				Edge 2					Edge *E*							CO 1					CO *C*							
1	2		*N*	1	2		*N*		1	2		*N*	1	2		*N*		1	2		*N*							
[*N*S* input traffic constraints in real variables]																											
[*N*E* edge pair constraints in real variables]																											
[*N* node capacity constraints in real variables]																											
[*N*C* installation cost constraints in mixed variables]																											

Table 6 1 LP/MIP Constraints Matrix

6 1 1 2 Algorithm Outputs

Having received a solution from the OSL Solver in the form of a row vector giving the network flow solution, some post-processing is done to extract the pertinent information The flows relating to each potential CO copy on the network nodes are examined If all flows are zero, then the CO is not assigned to that particular node The objects remaining give the solution CO allocation

The solution flows are also used to calculate random splitting probabilities between objects These are required to achieve the optimal flows in the network for the given user traffic mix To do this, the work-flows between a given CO copy and all copies of another CO in the network (which is part of the same service interaction) are examined As splitting probabilities are required only at points in service where a new CO type is needed to continue service execution, the splitting points of the solution flows must be interpreted in the order that the COs are encountered during a service session At these points, the solution flows between the source CO and target CO are normalised to give the set of random splitting probabilities

In practice, there are relatively few splitting points of interest because the problem solution tends to collocate objects as much as possible to reduce communications overhead Thus we

filter out all splits from the program output that *only* involve collocated COs

The solution flows for user traffic are transformed back to a *sessions per second* measure thus giving the arrival rates in the optimised network that produce maximum possible loading in the network, i e maximum throughput

6.1.2 Basic Results for Optimised CO Placements

We first examine the output of the optimisation for the following service platform specification

- There are 10 processing nodes all with relative processing speeds of 1
- Services, COs and messages are as per the test service MSCs and message details given in Chapter 4
- There are two sources (SSPs) connected to two gateway nodes
- Relative traffic volumes are all equal, i e each gateway receives the same traffic volume for each service and the total traffic at the two gateways is also equal

(Note that this specification is the same as that of the *Service Platform Model*, described in Chapter 4, in terms of number of processors, service MSCs and SSP connections)

The optimised output is shown in Tables 6 2(a) and 6 2(b) below Table 6 2(a) gives the optimal assignment of COs to nodes Note that the *GW*s have been fixed to nodes 1 and 2 (an arbitrary choice made in the input specification) SSPs are not shown as they are assigned to their own *GW* and cannot split traffic

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x		x			x			x			
2	x	x	x	x	x		x			x			x			
3		x	x	x	x		x			x			x			
4		x	x	x	x		x	x		x	x		x	x		x
5		x	x	x	x	x		x	x		x	x		x	x	x
6		x	x	x	x	x	x		x	x		x	x		x	
7		x	x	x	x		x	x		x	x		x	x		x
8		x	x	x	x			x			x			x		x
9		x	x	x	x	x	x		x	x		x	x		x	
10		x	x	x	x	x			x			x			x	

Tables 6 2 (a) Optimal CO Allocation Solution for Equal Arrival Rates all Services

Table 6 2(b) shows the optimal splitting between objects for each service For example, the first two rows for Service A show that the *GW* on Node 1 splits 0 2894 of its traffic with the *SCFP* on Node 9 and 0 7106 of its traffic with the *SCFP* on Node 10, for Service A traffic

By examining the splitting probabilities we see that all objects required for each service, apart from the *GW*s, have been grouped together with the *SCFP* and the group copied to all nodes This is evidenced by the fact that there are no other splittings, apart from *GW-SCFP* Also, groupings are collocated with the *GW*s as far as possible, to reduce communications time We note, however, that the *SCFP* group on node 1 receives only a relatively tiny amount of traffic but is still required by the optimal solution These low-utilisation objects can be eliminated from the solution by setting minimum CO utilities (discussed below)

Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	9	0 2894
	GW	SCFP	1	10	0 7106
	GW	SCFP	2	5	0 2857
	GW	SCFP	2	6	0 7143
B	GW	SCFP	1	1	0 0019
	GW	SCFP	1	6	0 0034
	GW	SCFP	1	7	0 5913
	GW	SCFP	1	9	0 4034
	GW	SCFP	2	2	0 0019
	GW	SCFP	2	3	0 6757
	GW	SCFP	2	4	0 3224
C	GW	SCFP	1	7	0 1111
	GW	SCFP	1	8	0 8889
	GW	SCFP	2	4	0 4649
	GW	SCFP	2	5	0 5351

Tables 6 2 (b) Optimal CO Allocation Solution for Equal Arrival Rates all Services

To verify the allocation, the service platform was simulated[2] using the optimal splitting probabilities given above These were employed as the *internal performance controller* splitting probabilities and are static throughout the simulation Thus we have a static internal control to compare to dynamic schemes (in later sections of this chapter) The traffic mix was also the same as at the design point – equal arrivals from all sources Note that there is no throttle implemented at the gateway (*External Performance Control*) in this simulation scenario Table 6 2(c) shows the results that were obtained from the simulator as follows

The throughput measure is a total over all arrivals to the network and was obtained by increasing the arrival rate until the load on the heaviest loaded processor averaged 90%

[2] Note that verification of the simulator and the methodology for assessing simulation results is discussed in §6 3 below Simulation results given here were obtained according to that methodology

(±1%) (In results obtained later, we set the throttle at 90% so aiming for this value here allows comparison) Note that the total throughput given is split between sources according to the relative traffic volumes specified for the optimisation, in this case equally

Throughput at Max Load=90%	122 (sessions s $^{-1}$)									
Installation Cost (1 cost unit per CO copy)	97									
Service Delays – Low Load (20% of Max)	93 (ms)									
Service Delays – High Load (90% of Max)	481 (ms)									
Processor % Utilisation (Nodes 1 to 10)	89 2	88 0	88 5	89 0	88 1	90 3	87 6	88 1	89 1	90 1

Table 6 2 (c) Performance of Optimal CO Allocations for Equal Arrival Rates all Services

Each CO copy allocation is deemed to cost 1 unit so the Installation Cost simply gives the total number of copies in the allocation Note that there was no limit set on the CO installation costs for the optimisation

Average service delays are measured for a low and a high arrival rate that nominally give 20% and 90% utilisation respectively The required arrival rate for high load is the same as that given for the throughput The required arrival rate for low load is calculated as

$$\lambda_{20} = \frac{20\%}{90\%} \rho_{90}$$

Where ρ_{90} is the 90% throughput Note that the *service delay* was measured as the total time a service session spends *in the service platform* It does not include processing or user interaction times at the SSP, which are not related to service platform performance To obtain the service delay, the total average SSP time is subtracted from the average session *Round Trip Time* measured by the simulator This gives a measure indicative of response times to a user's requests

Finally, the average utilisation of each processor is given in the last row of the table Note that the allocation has given full utilisation on all processors, balancing load Similar results given in the following sections were obtained as have been described here

6.1 3 Load Imbalance

Here we consider the allocations and splitting produced by the optimal placement algorithm when the service demand is not balanced between services We consider the traffic mix scenarios given below These values give the relative traffic mixes for input to the optimisation program All other inputs are the same as in the previous equal traffic mix scenario The results are given in Tables 6 3 to 6 5 below Note that, again, the optimal splitting probabilities were used in each case and the service mixes to the simulator were the same as the input traffic mixes to the optimisation program

	S1 GW1	S1 GW2	S2 GW1	S2 GW2	S3 GW1	S3 GW2
Scenario 1 Relative Arrival Rates	8	8	1	1	1	1
Scenario 2 Relative Arrival Rates	1	1	8	8	1	1
Scenario 3 Relative Arrival Rates	1	1	1	1	8	8

The results for Scenario 1 (Table 6 3) show that, as expected, the Service A COs have been replicated across nodes more than for Service B or C's B's COs are still quite heavily replicated as B is the most complex (and with the longest execution time) service B's COs are also collocated with the *GW* as much as possible as communication is heavier Again all processor loads are maximal A slightly higher throughput is attained due to the high proportion of Service A traffic which is a simpler service There is also a corresponding decrease in service times compared to the equal loading scenario

In Scenario 2 (Table 6 4), the high-load Service B has its arrivals increased As expected, Service B components are the most heavily replicated with service B specific COs on 7 remaining nodes The total throughput has decreased as the average service time is greater compared to the equal loading scenario Average service delays have also increased accordingly Again load is balanced across processors

Scenario 3 (Table 6 5) has increased arrivals for Service C, which is the service with the shortest execution time A higher average throughput and lower service delays are attained Note that Service B COs are still quite heavily replicated as it is the highest load service

The optimisation achieved load balance and thus full usage of available resources for all three loading scenarios Delays are reasonably low at high arrival rates in each case However, all three scenarios display a relatively large amount of component duplication Noting that there are 16 CO types in the network, on average there are approximately 5 5 copies of each component deployed This may be undesirable for cost and logistical reasons We next consider limiting the total installation costs of components to reduce replication

6.1 4 CO Installation Costs

The optimisation program allows specification of a maximum CO installation cost This cost is the CO installation cost multiplied by the number of its copies, summed over all COs Here, we examine the effect of reducing the maximum cost of the allocations produced In the experiments, we assume an installation cost of 1 unit for each CO type Thus, the resulting cost gives a count of the total number of CO copies installed Service traffic mixes are set equal and we compare with the results in Table 6 2 (which had the same offered traffic mix but with no installation cost constraint)

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x		x			x			x			
2	x	x	x	x	x		x			x			x			
3		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4		x	x	x	x	x			x			x			x	
5		x	x	x	x	x			x			x			x	
6		x	x	x	x	x			x			x			x	
7		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
8		x	x	x	x	x			x			x			x	
9		x	x	x	x	x			x			x			x	
10		x	x	x	x	x			x			x			x	

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	7	0 1363
	GW	SCFP	1	8	0 2935
	GW	SCFP	1	9	0 2935
	GW	SCFP	1	10	0 2767
	GW	SCFP	2	3	0 1196
	GW	SCFP	2	4	0 2935
	GW	SCFP	2	5	0 2935
	GW	SCFP	2	6	0 2935
B	GW	SCFP	1	1	0 4667
	GW	SCFP	1	3	0 0651
	GW	SCFP	1	7	0 4682
	GW	SCFP	2	2	0 4667
	GW	SCFP	2	3	0 5333
C	GW	SCFP	1	2	1 0000
	GW	SCFP	2	7	1 0000

(c) CO Distribution Performance

Throughput at Max Load=90%	128 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	94									
Service Delays – Low Load (20% of Max)	86 (ms)									
Service Delays – High Load (90% of Max)	431 (ms)									
Processor % Utilisation (Nodes 1 to 10)	91 6	89 2	89 3	90 5	88 1	88 2	89 3	88 0	89 4	87 9

Tables 6 3 (a,b,c) Optimal CO Allocations for Service A Arrival Rates = 8 x B = 8 x C (No installation cost limit and no minimum CO traffic limit.)

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x											
2	x	x	x	x	x											
3		x	x	x	x		x			x			x			
4		x	x	x	x		x			x			x			
5		x	x	x	x	x		x	x		x	x		x	x	x
6		x	x	x	x		x			x			x			
7		x	x	x	x		x			x			x			
8		x	x	x	x		x	x		x	x		x	x		x
9		x	x	x	x		x			x			x			
10		x	x	x	x		x			x			x			

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	5	1 0000
	GW	SCFP	2	5	1 0000
B	GW	SCFP	1	6	0 2963
	GW	SCFP	1	7	0 1850
	GW	SCFP	1	8	0 2224
	GW	SCFP	1	9	0 2963
	GW	SCFP	2	3	0 2963
	GW	SCFP	2	4	0 2963
	GW	SCFP	2	7	0 1111
	GW	SCFP	2	10	0 2963
C	GW	SCFP	1	5	1 0000
	GW	SCFP	2	5	0 1191
	GW	SCFP	2	8	0 8809

(c) CO Distribution Performance

Throughput at Max Load=90%	100 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	75									
Service Delays – Low Load (20% of Max)	109 (ms)									
Service Delays – High Load (90% of Max)	663 (ms)									
Processor % Utilisation (Nodes 1 to 10)	90 1	88 7	90 0	89 2	88 6	88 1	89 3	88 8	89 5	90 3

Tables 6 4 (a,b,c) Optimal CO Allocations for Service B Arrival Rates = 8 x A = 8 x C (No installation cost limit and no minimum CO traffic limit.)

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x		x			x			x			
2	x	x	x	x	x		x			x			x			
3		x	x	x	x		x	x		x	x		x	x		x
4		x	x	x	x		x	x		x	x		x	x		x
5		x	x	x	x			x			x			x		x
6		x	x	x	x	x		x	x		x	x		x	x	x
7		x	x	x	x		x	x		x	x		x	x		x
8		x	x	x	x			x			x			x		x
9		x	x	x	x			x			x			x		x
10		x	x	x	x	x		x	x		x	x		x	x	x

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	10	1 0000
	GW	SCFP	2	6	1 0000
B	GW	SCFP	1	1	0 4050
	GW	SCFP	1	3	0 0325
	GW	SCFP	1	7	0 5625
	GW	SCFP	2	2	0 4045
	GW	SCFP	2	4	0 5950
C	GW	SCFP	1	7	0 2276
	GW	SCFP	1	8	0 3016
	GW	SCFP	1	9	0 3016
	GW	SCFP	1	10	0 1692
	GW	SCFP	2	3	0 2973
	GW	SCFP	2	4	0 2233
	GW	SCFP	2	5	0 3016
	GW	SCFP	2	6	0 1778

(c) CO Distribution Performance

Throughput at Max Load=90%	141 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	97									
Service Delays – Low Load (20% of Max)	72 (ms)									
Service Delays – High Load (90% of Max)	361 (ms)									
Processor % Utilisation (Nodes 1 to 10)	88 6	88 1	88 0	87 3	88 7	90 6	87 9	89 1	90 2	88 7

Tables 6 5 (a,b,c) Optimal CO Allocations for Service C Arrival Rates = 8 x A = 8 x B (No installation cost limit and no minimum CO traffic limit.)

Table 6 6 (below) gives the results for a maximum installation cost of 60 units and a minimum CO utility value of 0 1 traffic units The installation cost limit has hampered the optimisation significantly and on average the network is only 77% utilised with Node 5 not used at all Some CO groupings have also split in two at the *SCFP-UAP* boundary This is explained by the reduced replication sought, as traffic must split to reach the more centralised COs For example, only one copy of the *UAP(B)* is allocated Even though the average throughput has decreased due to the lower processor utilisation, average delay has increased, as there is significantly more remote communication cost and more queues are encountered during a service session Note, however, that the low utility COs have been eliminated from the solution

As the above installation cost constraint resulted in effectively a 9-node network, it is interesting to compare this to an optimisation on a 9-node network (rather than 10) that has no installation cost constraint The resulting 9-node network gives a throughput of 110 sessions per second and a total installation cost of 78 That is, throughput has increased by approximately 17% but cost has increased by approximately 32% So, there may still be utility in the cost-constrained network, from a throughput/cost perspective Note that the scenario generally demonstrates the performance effects of disallowing duplication of COs Even a relatively small restriction in duplication can have a large effect on the maximum performance achievable

Figure 6 2 shows the trade-off between maximum achievable throughput and installation costs Note that there is a region (between CO costs of 78 to 97) over which costs may be reduced with little loss of throughput

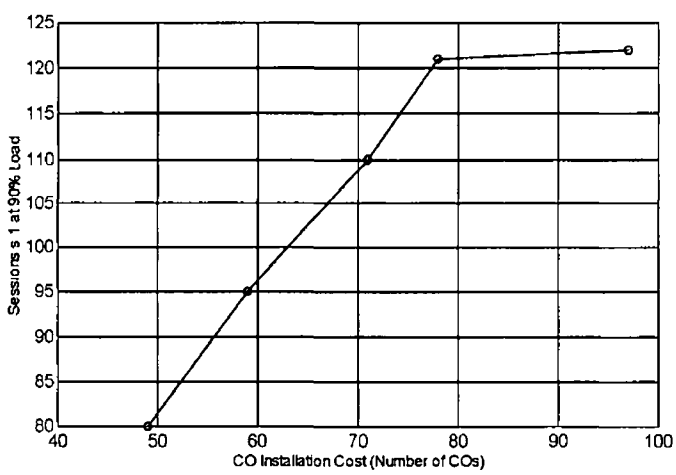


Figure 6 2 Reduction of Throughput with Reduced CO Installation Costs

We give the solution at the boundary point in Table 6 7 (cost of 78) and note that the performance is very similar to the original optimised network (Table 6 2) However, the

installation cost has been reduced from 97 to 78, a reduction of 19 6% We considered this to be the best trade-off when we are primarily concerned with throughput rather than cost Thus, this cost constrained CO placement has been used to complete the specification of the simulated Service Platform (Chapter 4) We will run dynamic random splitting schemes (rather than the thus far static ones) on this platform, which adjust to the incoming arrival rates These results are described in later sections of this chapter We will refer to the completed service platform with this CO allocation as the *reference platform*

As an observation, note that, it is difficult to relate the costs of CO installation to losses due to reduced throughput in a linear manner For this reason this has not been attempted in the optimisation formulation However, as illustrated above, installation costs can be used to reduce unnecessary CO copies without loss of performance and we propose this approach as a useful network design method

6.1.5. Scalability and Bottlenecking

We examine the effect of varying the allowable number of nodes in the network (Figure 6 3) Up to 12 nodes, there is a linear relationship between maximum achievable throughput and the number of nodes The linear optimisation approach ensures scalability, as throughput (and thus profitability) of the network scales linearly with the cost of deploying new processing nodes In a non-optimised network, the effect to increasing processing nodes may be difficult to predict Thus, the optimal approach may be a useful network dimensioning design tool

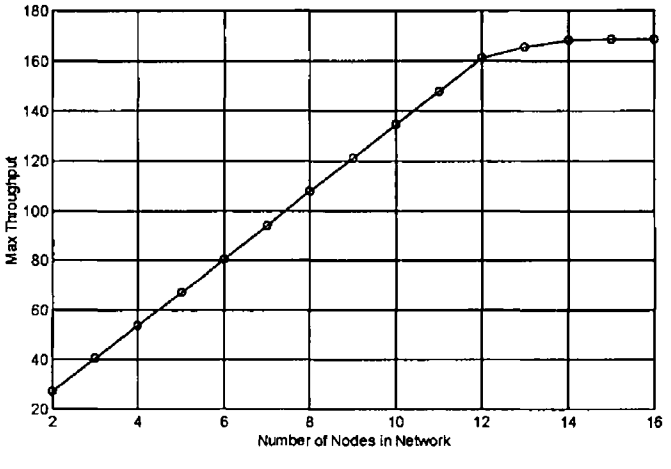


Figure 6 3 Increase in System Throughput as Processing Nodes are Added to the Network

Beyond 12 nodes the return on investment in processing power decreases This is due to the gateways starting to bottleneck the system beyond this point To gain more throughput, it would be necessary to split traffic across additional gateway nodes or to increase gateway processing power We can conclude however, that two IN/CORBA *GWs* running on generic nodes can still efficiently drive a network with 6 times their combined processing power

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x														
2	x	x														
3		x	x	x	x			x			x			x		x
4		x	x	x	x	x			x			x			x	
5																
6		x	x	x	x	x			x			x			x	
7		x	x	x	x	x			x			x			x	
8		x	x	x	x		x			x			x			
9		x	x	x	x			x			x			x		x
10		x	x	x	x			x			x			x		x

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	1	0 4002
	GW	SCFP	1	6	0 2534
	GW	SCFP	1	7	0 3464
	GW	SCFP	2	2	0 4002
	GW	SCFP	2	4	0 2958
	GW	SCFP	2	7	0 3040
	SCFP	UAP (A)	1	6	1 0000
	SCFP	UAP (A)	2	4	1 0000
B	SCFP	UAP (B)	1	8	1 0000
	SCFP	UAP (B)	2	8	1 0000
C	GW	SCFP	1	3	0 5942
	GW	SCFP	1	10	0 4058
	GW	SCFP	2	9	0 6449
	GW	SCFP	2	10	0 3551

(c) CO Distribution Performance

Throughput at Max Load=90%	94 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	59									
Service Delays – Low Load (20% of Max)	96 (ms)									
Service Delays – High Load (90% of Max)	510 (ms)									
Processor % Utilisation (Nodes 1 to 10)	87 1	89 8	76 0	90 2	0 0	89 0	88 5	74 5	88 3	90 3

Tables 6 6 (a,b,c) Optimal CO Allocations for Equal Arrival Rates Maximum Installation costs = 60 COs, with minimum CO traffic limit of 0 1

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	x	x		x			x			x			
2	x	x	x	x	x		x			x			x			
3		x	x	x	x	x			x			x			x	
4		x	x	x	x		x			x			x			
5		x	x	x	x			x			x			x		x
6		x	x	x	x			x			x			x		x
7		x	x	x	x	x			x			x			x	
8		x	x	x	x			x			x			x		x
9		x	x	x	x	x			x			x			x	
10		x	x	x	x		x			x			x			

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	3	0 6434
	GW	SCFP	1	7	0 3566
	GW	SCFP	2	7	0 3496
	GW	SCFP	2	9	0 6504
B	GW	SCFP	1	1	0 2561
	GW	SCFP	1	4	0 7439
	GW	SCFP	2	2	0 2561
	GW	SCFP	2	10	0 7439
C	GW	SCFP	1	5	0 3137
	GW	SCFP	1	6	0 6863
	GW	SCFP	2	5	0 3726
	GW	SCFP	2	8	0 6274

(c) CO Distribution Performance

Throughput at Max Load=90%	121 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	78									
Service Delays – Low Load (20% of Max)	91 (ms)									
Service Delays – High Load (90% of Max)	472 (ms)									
Processor % Utilisation (Nodes 1 to 10)	88 8	89 2	90 0	88 6	89 0	90 4	89 1	90 3	90 1	88 5

Tables 6 7 (a,b,c) Optimal CO Allocations for Equal Arrival Rates Maximum Installation costs = 80 COs, with minimum CO traffic limit of 0 1

In conclusion, we have seen that the throughput optimisation tends to collocate objects as much as possible, which will tend to reduce the number of queues that the service messages encounter (as well as reducing processing due to encoding/decoding) Thus, although the objective is to maximise throughput a desirable side effect is that delays are kept at reasonable levels Also, in each scenario examined, loading has been balanced across processors

6.2. Comparison to Minimisation of Communications Costs

The majority of current optimal allocation problems aim to minimise communications costs rather than maximising throughput (for example [Anagnostou, 1998], [Bastarrica *et al* 1998] and [Avramopoulos & Anagnostou, 2002]) We apply a communications minimisation objective to the model and examine differences in results to our throughput maximisation approach

Using the notation already defined in §5 1 we define the communications cost, ${}^{s}_{m}C^{n}_{ij}$, as the total ORB encoding/decoding time on processor n and processor m due to all communication between an instance of computational object c_i on processor n and an instance of computational object c_j on processor m during one service session of type s

$$ {}^{s}_{m}C^{n}_{ij} = \begin{cases} \sum_{\forall k \in {}^{s}M_{ij}^{ci\,(client)}} \tau_n^{encode}(k) + \sum_{\forall k \in {}^{s}M_{ij}^{cj\,(server)}} \tau_n^{decode}(k) & \forall\, n \neq m \\ 0 & \forall\, n = m \end{cases} $$

Thus ${}^{s}_{m}C^{n}_{ij}$ gives a communications costs in terms of the same encoding/decoding times used in the throughput maximisation model, allowing a fair comparison between the problem solutions The problem objective is

$$ \textit{Minimise} \qquad \sum_{\forall {}^{s}x_{ij}^{nm}} {}^{s}_{m}C^{n}_{ij}\ {}^{s}x_{ij}^{nm} $$

i e to minimise all times relating only to remote communications costs between COs The flow balance, processor limit and user location constraints are as defined for the original throughput optimisation (§5 1 4) In this case, the user demands are fixed values (otherwise the solution would tend to zero) This constraint replaces the relative arrival rate constraints of the original throughput problem and may be stated as

$$ \Lambda_s^n = \overline{\lambda}_s^n \qquad \forall n \in N_{user}, s \in S $$

where $\overline{\lambda}_s^n$ are the assumed arrival rates for the system A solution to an instance of this problem is shown in Tables 6 8 (a) and (b) The solution was obtained for the following set of arrival rates $\{\overline{\lambda}_s^n\}$

	Service A	Service B	Service C
GW (Node 1)	4 (session s^{-1})	32 (session s^{-1})	4 (session s^{-1})
GW (Node 2)	4 (session s^{-1})	32 (session s^{-1})	4 (session s^{-1})

(a) Computational Object Distribution

Node	GW	SCF Proxy	PA	UA	GSEP	UAP (A)	UAP (B)	UAP (C)	SF (A)	SF (B)	SF (C)	SSM (A)	SSM (B)	SSM (C)	SSO (A)	SSO (C)
1	x	x	x	X	x		x			x			x			
2	x	x	x	X	x		x			x			x			
3		x	x	X	x		x			x			x			
4		x	x	X	x		x			x			x			
5		x	x	X	x	x	x	x	x	x	x	x	x	x	x	x
6		x	x	X	x	x	x	x	x	x	x	x	x	x	x	x
7		x	x	X	x	x	x		x	x		x	x		x	
8		x	x	X	x		x			x			x			
9		x	x	X	x		x			x			x			
10																

(b) Random Splitting Probabilities

Service	Source CO	Target CO	Source Node	Target Node	Splitting Probability
A	GW	SCFP	1	7	1 0000
	GW	SCFP	2	5	0 6696
	GW	SCFP	2	6	0 3304
B	GW	SCFP	1	1	0 0023
	GW	SCFP	1	3	0 4169
	GW	SCFP	1	4	0 4169
	GW	SCFP	1	5	0 1640
	GW	SCFP	2	2	0 0023
	GW	SCFP	2	6	0 2220
	GW	SCFP	2	7	0 2456
	GW	SCFP	2	8	0 4169
	GW	SCFP	2	9	0 1131
C	GW	SCFP	1	6	1 0000
	GW	SCFP	2	5	1 0000

(c) CO Distribution Performance

Throughput at Max Load=90%	76 (sessions s^{-1})									
Installation Cost (1 cost unit per CO)	85									
Service Delays – Low Load (20% of Max)	112 (ms)									
Service Delays – High Load (90% of Max)	682 (ms)									
Processor % Utilisation (Nodes 1 to 10)	89 4	88 0	90 6	88 3	88 4	88 1	89 4	88 8	22 0	0 0

Tables 6 8 (a,b,c) Optimal CO Allocations for Communications Cost Minimisation

Note that the traffic mix is the same as for results Table 6 4 where Service B traffic is eight times heavier than A or C The total traffic intensity was chosen to give a maximum processor load of approximately 90% The results show (Table 6 8) that the minimum communication cost optimisation is not driven to use all processing capacity, as throughput maximisation is

The minimum communications cost solution will not balance load unless the arrival intensity causes loading on all nodes to be maximal Simulation results are given in Table 6 8 (c) Node 10 has no COs allocated at all and Node 9 is only loaded to 22% As expected, Service B is the most heavily replicated with COs on 9 remaining nodes Apart from load-balance, the solution displays similar properties in terms of object grouping as the throughput maximisation solution (Table 6 4)

There are potential setbacks to this approach As loading on nodes will not necessarily be balanced in the solution, there is a larger variation in delay when a service may be processed on either a high or low loaded node Also, on average it is likely that the delay will be greater than if the same amount of load is shared equally amongst nodes (as delay is generally an increasing function of load) Also, as too high arrival intensities will cause an infeasible problem space, it is not suited to admission control optimisation Throughput maximisation thus has several advantages

Regarding application to dynamic sharing algorithms, as nodes may be loaded to 90% even when the system is relatively lightly loaded, there is no 'head-room' to accommodate sudden increases in traffic This is especially important for dynamic controls, which may not update random splitting information fast enough to cope with transients Thus, with communications minimisation solutions, the processor may be in danger of overload even under low system load conditions

6.3. Simulation Methodology and Validation with an Analytic Model

We make a note here regarding simulation methodology before examining further results for the dynamic internal and external controls In order to gain accurate simulation results for the service platform we have used the following methods for ensuring high confidence

Simulations are run until the customer population in the system has stabilised and the output measure we require has reached close to its final value The output trace for the required measure is examined and the average values over each of two consecutive periods at the end of the simulation run are compared The length of each of these intervals is 5% of the total simulation run time The averages are required to be within 2 % of each other to indicate that the simulation has settled If not, the simulation run is elongated until the settling condition has been reached To ensure confidence in the results, a number of simulation runs are executed in accordance with the steady state condition The number of simulation runs is increased until not worse than 95% confidence intervals of ±2% are achieved in the measure of interest In order to validate the functioning of the simulator, we compare results to those obtained with an analytic model, described below

6.3.1. Analytic Model of the Service Platform

In this section a *Layered Queuing Network* (LQN) model is developed to obtain an analytic solution for mean processor loads and mean service execution delays for the *reference platform* The objective is to verify the simulation model with analytic results The *Layered Queuing Network Solver* (LQNS) has been used to obtain solutions to the LQN Layered Queuing Networks and the LQNS have been described in §3 1 9

6 3 1 1 Model Assumptions

The model is based on the CO allocation of the *reference platform* (Table 6 7 (a)) There are four main distributable object types on the platform, the *GW* and the group of collocated service components for each of the three services For example, the collocated components for Service A are {SCF Proxy, PA, UA, GSEP, UAP(A), SF(A), SSM(A), SSO(A)} This grouped service-specific object is referred to here simply as the *SCF* The *SCF* encapsulates all COs that are grouped together and the execution times for message calls on its interface are based on local communication times between the encapsulated COs Messages received by an SCF are processed as a single message, where the processing time takes into account only local processing between grouped COs It is assumed that messages between COs occur as native function calls within the same thread of execution

Firstly, a general model is developed which allows two fixed-location *GW* objects and *SCF* copies to be placed on any of the network nodes This model is then used to obtain a solution to the specific *SCF* placements and random splitting specified by the reference platform A model for a single service type is first developed This is then extended to a multi-service network

All assumptions made for the service platform model (Chapter 4) also hold for the analytic model In summary

- There is one queue and one server on each of 10 processing nodes in the network
- Messages are asynchronous i e the calling process does not block waiting for a reply
- Arrivals and user interaction times at the SSP are exponentially distributed
- Service times are deterministic, as given in the message detail charts in Chapter 4

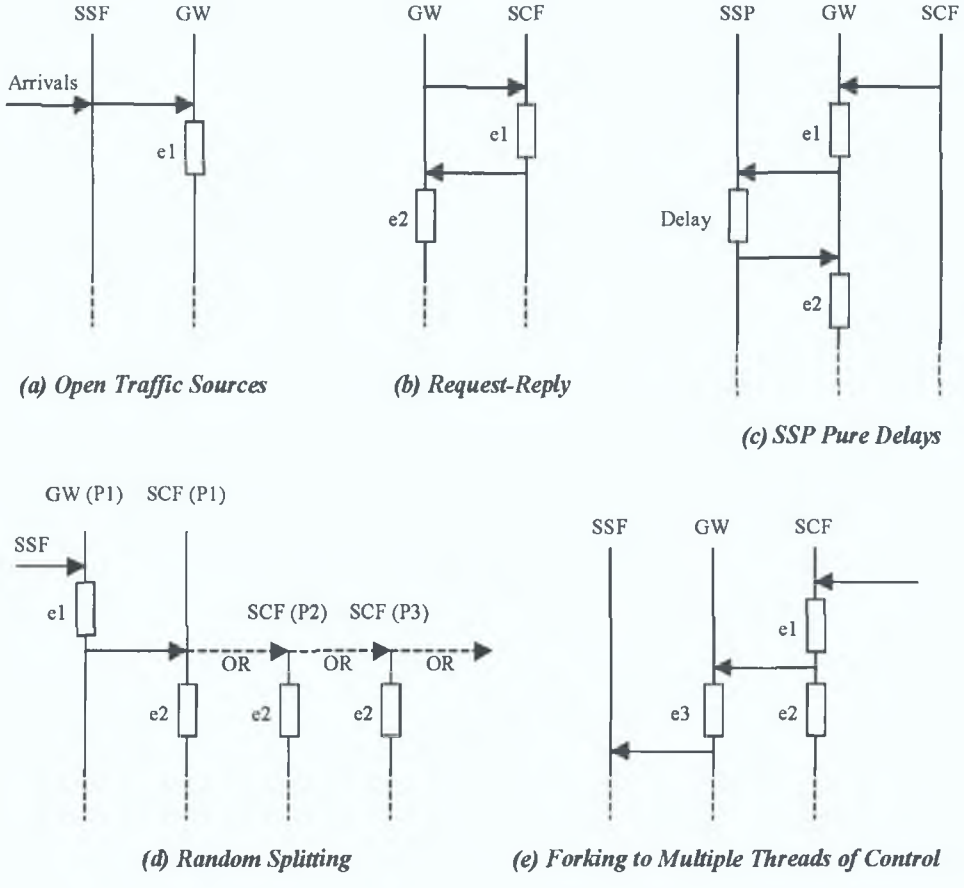
6 3 1 2 Execution Patterns

We may decompose the service MSCs into a number of specific execution patterns so that the model may be built-up from smaller, simpler modules Figure 6 4 below shows the set of five interactions that make up any of the three services on the reference platform They are as follows

Open Traffic Sources: We require six independent Poisson traffic sources to drive the model, one for each gateway/service combination. The mean arrival rate of each source is variable.

Request-Reply: The main interaction pattern in the MSC we wish to model is an asynchronous request-reply. Referring to Figure 6.4(b), a call sent form the *GW* to the *SCF* returns immediately (does not block the GW task). Message *e1* is queued and eventually executed on the *SCFs* processor and a message returned, again, freeing the *SCFs* processor as soon as the message is sent.

SSP Pure Delays: We do not model queuing in the Intelligent Network. However, we do wish to model user interaction times (e.g. conversion time during a call) and also take account of delay due to other message processing at the SSP (i.e. the CTR/ARI message pair in Services 2 and 3). These delays may be modelled as an infinite server with exponentially distributed service times, i.e. a pure delay.



(a) Open Traffic Sources *(b) Request-Reply*

(c) SSP Pure Delays

(d) Random Splitting *(e) Forking to Multiple Threads of Control*

Figure 6.4(a-e): Service Execution Patterns

Random Splitting: In order to model the internal performance control, we require a mechanism to model the random distribution of incoming service requests. According to the optimal splitting solutions, this is done by splitting the request stream at the *GW* to multiple *SCFs* (Figure 6.4(d)).

Concurrent Execution At some points in the MSCs a service continues execution in parallel For example, referring to Figure 6 4(e), having processed message *e1*, the *SCF* makes a call on entry *e3* of the *GW* task, but then immediately continues processing entry *e2* on its own processor

We derive each of these model elements below and then construct the overall system model of the service platform from the constituent parts

6 3 1 3 Modelling Open Traffic Sources

We require a LQN sub-model for the MSC of Figure 6 5(a) External arrivals to task T1 are generated by a Poisson process with an inter-arrival rate of λ The MSC can be expressed as an open arrival LQN model as in Figure 6 5(b) However, the LQNS analytic solver is primarily oriented towards solving queuing models with finite customer populations whereas our system is open In this case, the method given in [Shousha *et al*, 1998] is used to convert open models to closed In this method, the open arrival is replaced with N 'pure' client tasks which each cycle continuously (i e 'arrive to the system') on average Z times per second If N is very large, a value of Z may be chosen (appropriately large) such that an effectively open arrival process of rate $\lambda \approx N/Z$ is achieved That is, we approximate an infinite population with a very large one Note that, the clients must make *synchronous* (*rendezvous*) calls on task *T1*, so that they receive a reply and return to the client pool, ready to 'arrive' again Otherwise the client pool would eventually exhaust

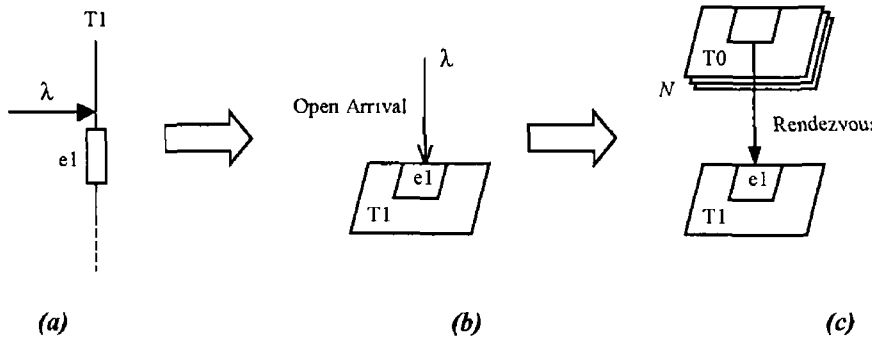


Figure 6 5(a-c) Conversion of Open to Closed Arrivals in the LQN

6 3 1 4 Modelling Requests-Replies and Message Execution

In order to model the interaction pattern of Figure 6 6(a), we note that an LQN must be specified as an *acyclic* task graphs This is to prevent *deadlock* occurring However, the service MSCs are cyclic in nature – a call is made to an object, which processes and then calls back to the original caller The transformation to an acyclic LQN is shown within the dashed box of Figure 6 6(b) Each message in the original sequence is modelled as a separate 'pseudo-task' with one entry corresponding to the call These tasks have no associated

processor and their entries have zero delay Calls are chained together with rendezvous calls, so that each message must finish processing before the next begins

To represent the resource aspects of the service, tasks with associated processors are required In the sub-model shown, the GW object is assigned to one processor (*P1*) and the SCF to another (*P2*) Their associated tasks, *P1 exec* and *P2 exec* allow modelling of the resource demands of calls on the GW and SCF When an entry of a pseudo-task is called, a rendezvous call is immediately made on the corresponding entry of the processor task This call blocks until the entry has queued and been executed On return, the task then continues with a call to the next pseudo-task, which represents the next message in the sequence And so the chain continues, blocking and waiting for execution of each message before continuing Note that all message pseudo-tasks in the chain behave as if they are 'infinitely threaded' That is, any number of messages can be in the system and blocked-waiting for access to the processors These calls are all queued and executed in FIFO order

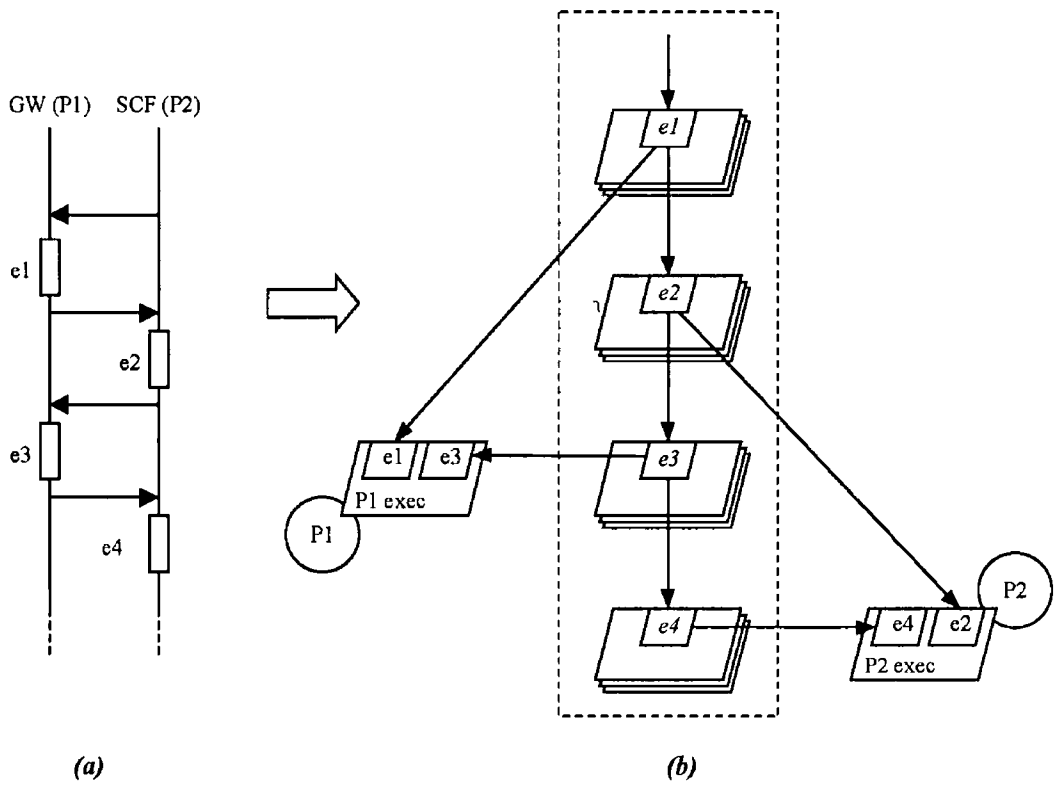


Figure 6 6(a,b) Modelling Requests-Replies and Message Execution

Note also that, as all calls are rendezvous, the original call from the client pool is blocked until the end of the message chain (after *e4* executes in this case) At this point it is returned to the client pool Note that there is only one (single threaded) task per processor so that each message is executed sequentially (Multiple tasks on a processor would share processor time by time-slicing, which is not our required behaviour) Thus the processor task must have an entry, with appropriate workload parameters, for each message that can be executed on the

processor Thus the model gives the required execution semantics That is, (1) calls do not block the calling process as it is threaded This, effectively, gives asynchronous call behaviour (2) Each processor is a single process with one FIFO message queue (3) The required sequence of calls and their execution is maintained (4) Each message has it own execution time (modelled as a deterministic service time according to message times given in the appendix to Chapter 4)

6 3 1 5 Modelling Random Splitting

We require a model for the random splitting that occurs at the gateway, that is, creation of a new thread of execution on a different processor, chosen according to a set of splitting probabilities The MSC pattern, for a three-processor system, is shown in Figure 6 7 and the corresponding transformation to an LQN sub-model is shown in Figure 6 8 (Note that, in Figure 6 8, processor tasks and all their entries are simply represented by their corresponding processor, a labelled circle)

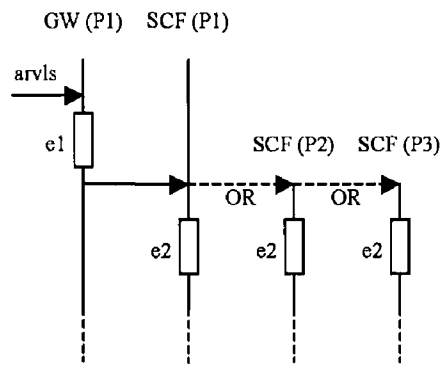


Figure 6 7 Random Splitting to Three Processors

An *activity* model element with *branching point* is used to perform the splitting With this construct one of the paths is chosen according to the set of probabilities $\{p1, p2, p3\}$ (In practice, there will be a probability associated with each potential host) Note that the branching task has no delays or associated processor and does not make calls to the processor tasks It merely performs the splitting to different message chains Having chosen a branch, the relevant message chain is started with a call to entry *e1* Each message chain then executes the service either on the GW processor (*P1*) or, the GW processor and one other processor (either of *P2* or *P3*) In the case of local processing on *P1*, the GW and SCF objects are amalgamated (conceptually) into one distributed object This object has consecutive messages grouped into single messages, where each message ends on a call to the SSF (not shown in Figure 6 8) or end of service Otherwise, the original messages in the MSC would be modelled as exiting and re-entering *P1* via its queue For this reason, the splitting is decided *before* execution of *e1*

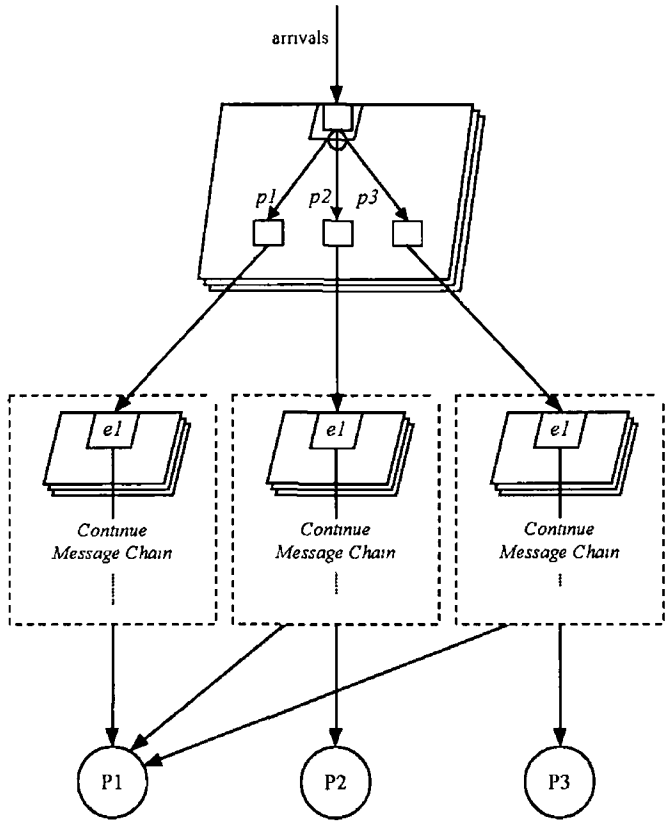


Figure 6 8 LQN for Random Splitting to Three Processors

6 3 1 6 Modelling SSP Delays

We do not consider queuing in the SSP and model calls to the *SSF* as pure delays This is modelled with the LQN *infinite server* construct Calls made to it are accepted immediately and block for a negative exponentially distributed time period and then return The server task has one entry for each delay required (e g User Interaction B1 requires a mean delay of 5 seconds) The delay tasks are employed similarly to ordinary processor tasks in the model

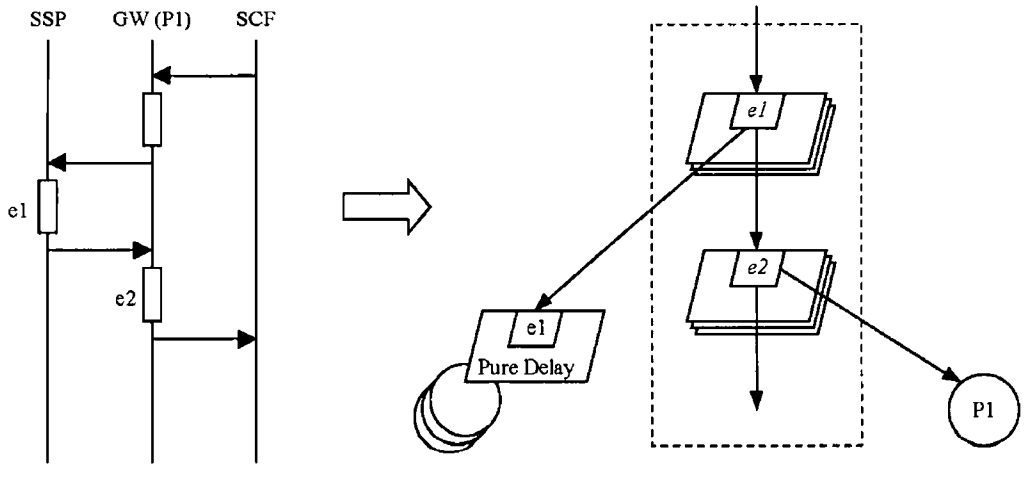


Figure 6 9 Delays Modelled as LQN Infinite Servers

6.3.1.7 Modelling Concurrent Execution

Parallel execution occurs when a message is sent to another processor but the sending thread then continues its own execution. An example of this pattern and its corresponding LQN is shown in Figure 6.10. (The example shown occurs at the end of each service session. Similar patterns are treated in the same manner.) An inter-task Fork-Join interaction (see [Franks, 1999]) has been used to start concurrent execution of entries *e2* on processor *P2* and *e3* on processor *P1*. The last task does not complete until both e3 and e2 have called it. Note that, again tasks in the dashed box are pseudo-tasks and have zero delay.

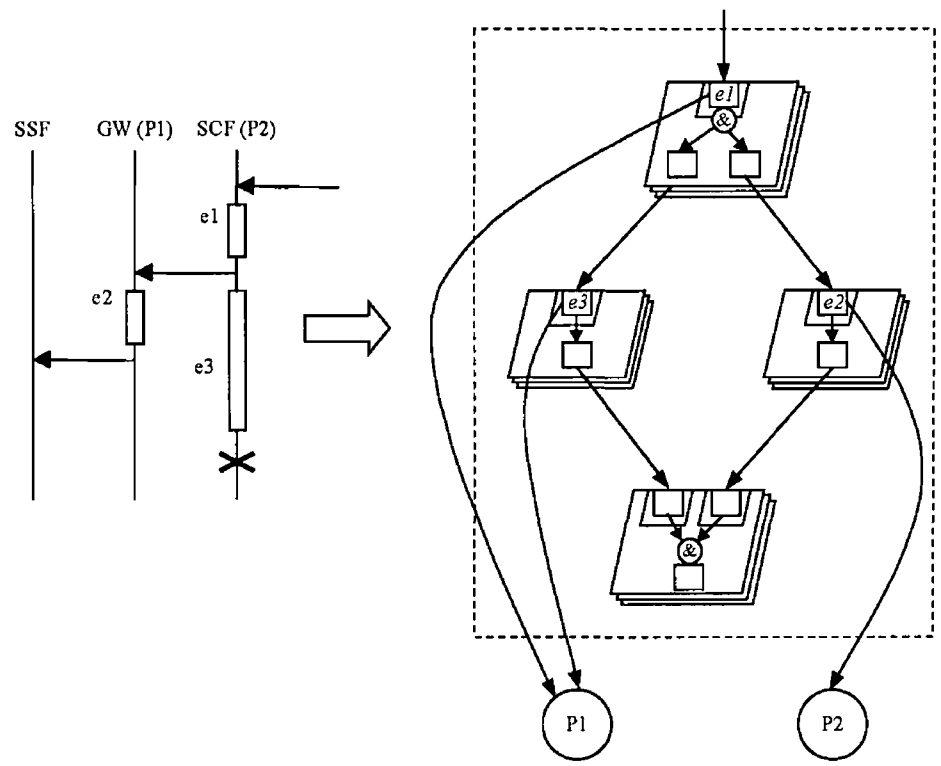


Figure 6.10 Modelling Deterministic Parallel Execution

6.3.1.8 Modelling Multiple Service Types and Overall Model

The overall model is constructed from the sub-models discussed above. The general model (that allows splitting to all 10 processors) is shown in Figure 6.11. Six separate traffic sources are required, one for each (GW, Service) pair. Each (GW, Service) source drives a separate message sequence module, composed of a branching point, which splits to one of 10 message sequence chains (*s1* to *s10*). Each of these sequences execute on the GW processor and (at most) one other processor corresponding to the sequence number. For example, sequence *s10* executes on *P10* and the GW processor *GW1* is deemed (arbitrarily) to execute on *P1* and *GW1* on *P2*. Thus, for example, *s1* of the left-most module only executes on *P1* (but may also call the pure delay server D1). All other sequences in this module execute on *P1* and their correspondingly numbered processor. The server pool contains one processor task for each

processor and a generic pure delay task which has entries for all required delay times for all services

Note that this is the general case For modelling the optimal allocations (which give random splittings to only a few SCFs from each GW), only a fraction of the sequences and processor task entries are required Note also that the model allows processors to be heterogeneous in the sense that each processor has independent processing times for a particular service message However, in our experiments, times are set equal for a given message across all processors, as was the case for the simulations

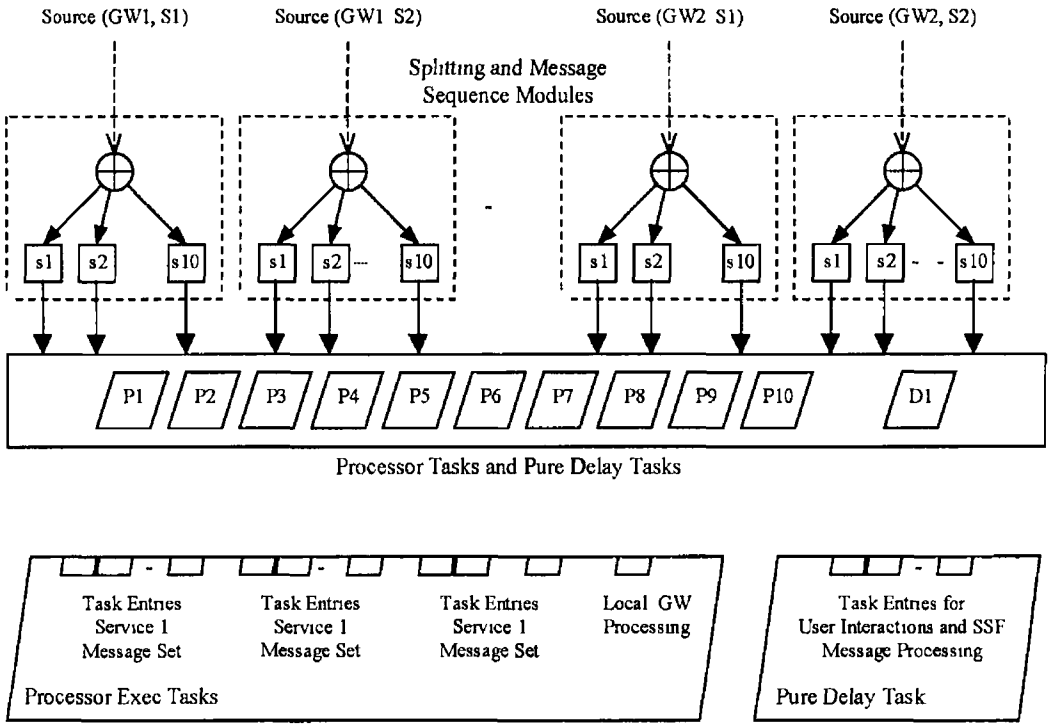


Figure 6 11 Overall LQN Model

6 3 2 Verification of Simulator with Analytic Solutions

To verify the operation of the simulator, an LQN model was constructed for a system with equal arrival rates for all services The object placements and random splitting probabilities are as given in Table 6 7 This scenario was also simulated and the results for average processor loading and service session delays compared to the analytic results Comparison of system loading is given in Figure 6 12 There is good correlation between the simulation and analytic solution over the range Figure 6 13 compares total service delays in the system There is good correlation between the two with a small discrepancy in the delay values at higher arrival rates Note that, due to the approximate nature of the analytic solution method, some variation between simulation and analytic results is expected The simulation results are expected to be most accurate

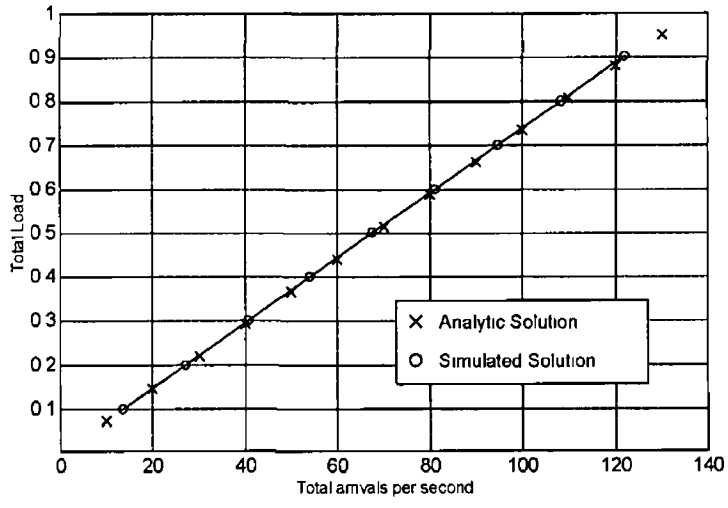


Figure 6 12 Simulated Versus Analytic Processor Utilisation

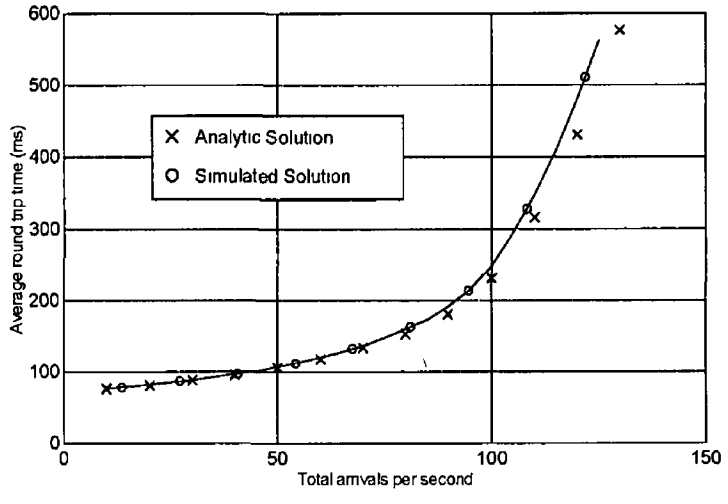


Figure 6 13 Simulated Versus Analytic Average Service Times

6.4. Performance of Dynamic Controls

In §6 2 we have considered the optimal static placement of COs on network nodes and explored the related performance issues by simulating at the design point with the optimal random splitting as an internal performance control In this section, we consider optimal internal and external dynamic controls We compare our controls to a simple intuitive load balancing mechanism to assess the merits of optimising the controls

6.4.1. Internal Performance Control

Thus far, we have not considered performance when service traffic mixes vary from the optimal design point To illustrate this scenario, consider the simulation traces in Figure 6 14 below Shown are the loads on the most and least loaded nodes in the network over time, where the traffic mix changes from the design point in the central region of the graph The network has been optimised for equal arrival rates for all services and is running using the resulting optimal random splitting probabilities as an internal control Service arrival rates are 20 sessions per second for each service At time t=500s the demand for Service B doubles and remains at this rate for a period It can be seen from the graph that once the service mix is changed the random splitting fails to perform optimally Indeed the highly loaded node is in danger of overload even though the average network load is relatively low (about 60% in this instance)

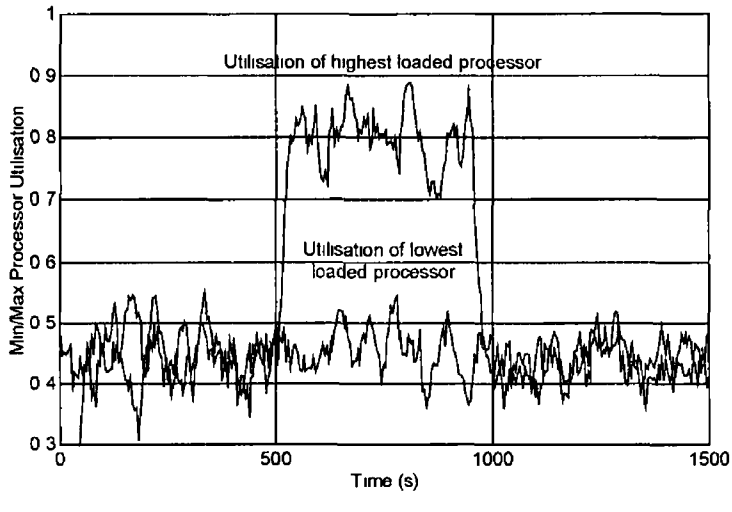


Figure 6 14 Loading when Traffic Mix Changes and Random Splitting is Fixed

In §5 1 6 we have proposed applying an LP to periodically update the random splitting probabilities given the arrival rates in the previous control period This problem is similar to the previous optimisation problem with the exception that the COs are now fixed and the desired traffic mix is estimated from a measure of the arrival intensities over the previous control interval, rather than being fixed arbitrarily at design time The objective is still to

maximise throughput and a new set of splitting probabilities is produced at the end of each control period, which should be optimal for the current traffic

Figure 6 15 shows the results of employing this dynamic control to the same scenario as before Shown are the loads on the same two nodes Although, the load has again increased due to the increase in Service B arrivals, load is quite evenly distributed In fact, load on all 10 nodes over the high load period was measured to be within 3% of the average

In the next section, we consider the broader performance properties of the internal control when coupled with the external control

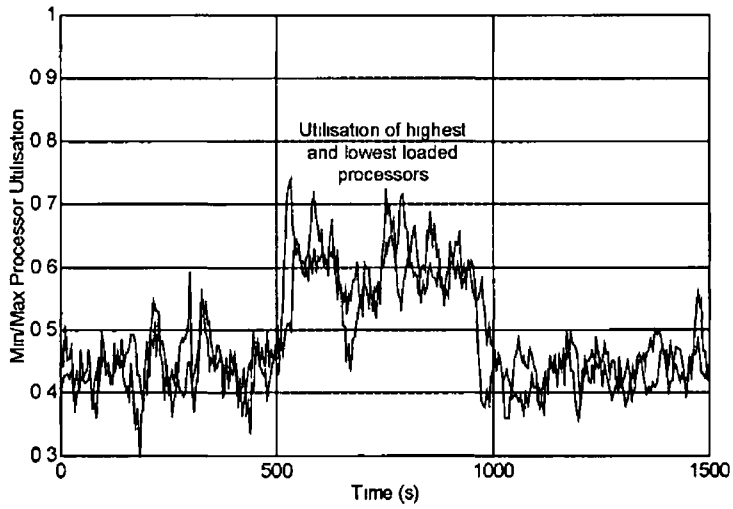


Figure 6 15 Loading when Traffic Mix Changes and Random Splitting is Dynamic

6 4 2 Internal and External Controls

Part of the solution to the LP in §5 1 6 is a set of maximum session arrival rate thresholds for the network In order to maintain the system at the optimum, it is necessary to limit the service request volumes at the gateway when arrival rates exceed these thresholds (the external control) When coupled with the internal control, the aim of the algorithm is to limit the load on all nodes in the network and to maximise throughput

The LP solution returned at the end of the control period contains both the random splitting probabilities, discussed above, and a set of acceptance probabilities for new arrivals at the gateways These probabilities are used by the Percentage Thinning algorithm (described in §5 2 5) to decide when to reject messages Note that only the first message of the service session may be rejected as otherwise processing time would be lost in the network due to sessions being aborted by the controller midway through execution

We use the *reference platform*, which has been optimised for equal service mix, to examine the combined internal and external controls However, in order to test the controls operating

away from the design point, we change the service mix so that there is twice the average arrival rate from Service B as from Service A or C We have chosen Service B as the dominant service as the CO allocation results have shown that this service is the most difficult to optimise, due to its longer service processing times

The simulation setup for the experiments that follow is

- Both dynamic internal and external controls are run on the reference platform setup
- The control periods are 20 seconds long
- Control information is disseminated instantaneously at the end of the control period

The relative arrival rates of each traffic source are as follows

Service A SSP1	Service B SSP1	Service C SSP1	Service A SSP2	Service B SSP2	Service C SSP2
1	2	1	1	2	1

The arrival rates are constant over each simulation run and the maximum processor utilisation is 90% Load throttles are implemented by Percentage Thinning

Figures 6 16 and 6 17 give the simulated processor utilisation and the service delay over a range of arrival rates for the internal and external controls operating together Note that the arrival rates given are totals over all traffic sources and the total load figure applies to the system as a whole (i e it is equivalent to average processor utilisation over all 10 nodes) From the processor utilisation, it can be seen that the throttles achieve close to the target of 90% load and that they are stable in the throttling region Note that all processors individually had utilisation within 1 7% of the 90% limit at an arrival rate of 150 sessions per second From the service session delay time, it can be seen that delay is bounded by the throttle and that this upper bound is reasonably short compared to the minimum delay

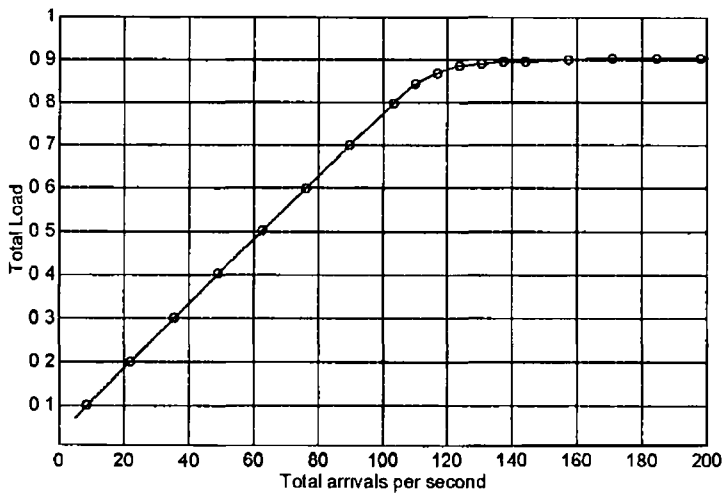


Figure 6 16 Total System Load with Internal and External Controls in Operation

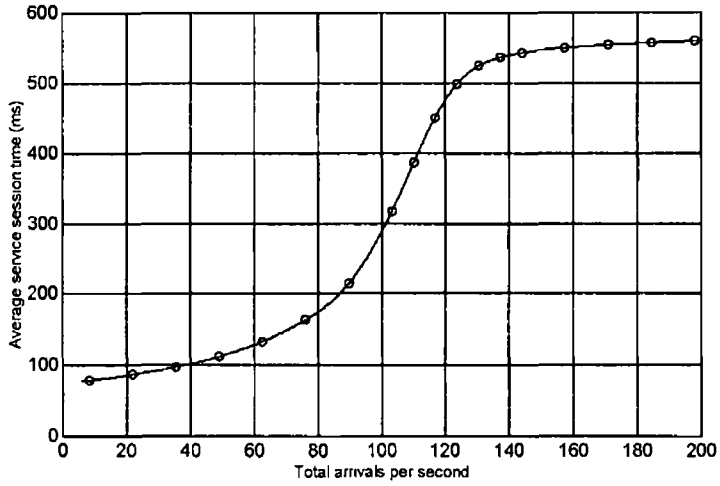


Figure 6 17 Average Service Delays with Internal and External Controls in Operation

To assess the merits of the optimal scheme we compare it to a simple load-balancing algorithm This algorithm operates by calculating the random splitting probabilities in proportion to the remaining processor utilisation, which is sent from all processors, every 20 seconds The splitting probabilities and the acceptance probabilities for admission control are calculated as

$$ {}^{s}_{split}p^{nm}_{ij}(k) = \frac{1-\rho^{m}(k-1)}{\sum_{\forall n \in N}\left(1-\rho^{n}(k-1)\right)} $$

$$ {}_{acpt}p^{n}_{s}(k) = \begin{cases} \dfrac{\rho^{n}_{\max}}{\rho^{n}(k-1)} & \rho^{n}_{\max} < \rho^{n}(k-1) \\ 1 & \rho^{n}_{\max} \geq \rho^{n}(k-1) \end{cases} \qquad \forall n \in N_{user}, \forall s \in S $$

where $\rho^{n}(k-1)$ is actual processor utilisation measured over the previous control interval and where α is a damping coefficient that allows damping of the response to rapid changes in processor load between control periods $\rho^{n}_{\max}$ is the load limit on gateway node n and is set to a value of 0 9 The scheme was implemented to adhere to the CO locations of the reference platform and operated over the same 20 second control period as the optimal algorithm Acceptance probabilities for the Percentage Thinning are calculated from the acceptance probabilities as per usual The value of α was tuned to achieve a stable control but was not otherwise found to affect the results to a significant extent

The comparison between the loading and delay for the optimal and simple load-balancing is given in Figures 6 18 and 6 19 The simple algorithm performs poorly as an internal control, as there are too many distributed requests made, which incur additional encoding/decoding processing times This increases load and delay Also, there are a larger number of queues

encountered during a service session, service session times become very long at high arrival rates The simple algorithm was effective in limiting load at its 90% threshold value and individual processors were well load-balanced (within 2%) However, this comparison illustrates that it is not enough to aim only for load balance and to prevent overloads The complexity of the system requires more than a simple greedy approach

6 4 3 Dynamic Performance Controls - Implementation Note

Much of the implementation of the dynamic performance controls is similar to that of the optimal CO allocation program, described in §6 1 However, the LP program needs to be called from the simulation rather than from the command line To enable this, our code is compiled with the OSL Solver headers to an object rather than an executable and an OPNET executable is made which links this object and the OSL Solver library During the simulation, the simulation calls the optimal controller code (sending the current arrival rate estimates) at the prescribed intervals to retrieve the new set of splitting probabilities with which it updates its load balancing lookup tables Note that this all occurs in zero simulation time All static data such as the number of nodes, COs and workload details etc is hard coded into the controller for efficiency The controller also returns the desired values of the arrival rates, which the external load controller uses to throttle traffic

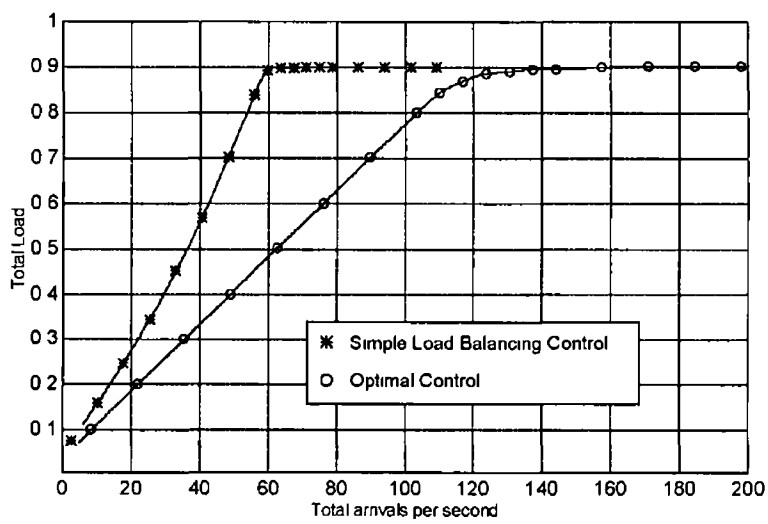


Figure 6 18 Reduced Throughput of Simple Load-Balancing Algorithm

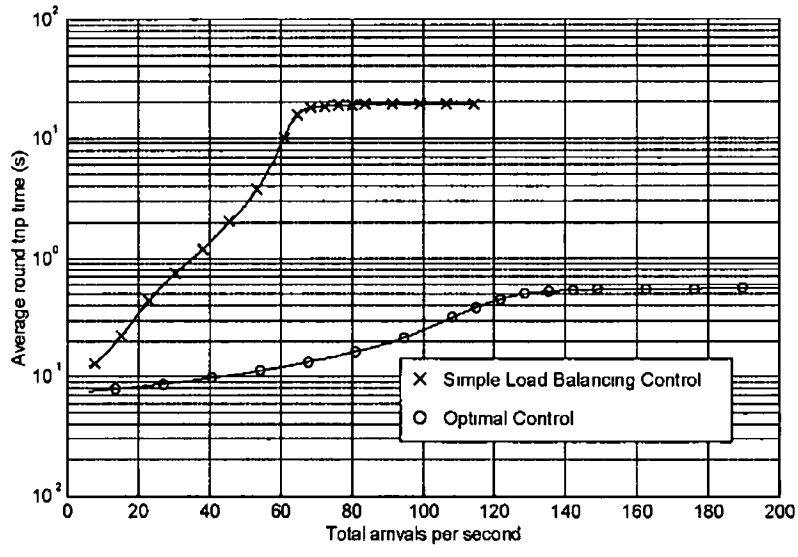


Figure 6 19 Increase in Service Delays of Simple Load-Balancing

6.5. Revenue Optimisation and Fairness

Throughput optimisation is extended to optimisation of network revenue with the algorithm proposed in §5 1 8 The aim is to consider the profitability of successful service sessions for the operator where each service type generates different revenues Rather than throttling all service types equally during periods of high arrival intensities, services are throttled differentially to achieve a higher overall utility for the network In the algorithm, profitability is offset by a simple 'fairness' coefficient, which may be set to limit the maximum attainable revenues but giving a fairer treatment across different service types during throttling

Figure 6 20 shows the total system revenue during throttling as arrivals to the network increase Relative arrival rates for all services are equal and service types are assigned the following revenues, in arbitrary units of revenue Service A 7 units, Service B 1 unit and Service C 3 units All other aspects of the system are as per the reference service platform The system revenues have been obtained for three different fairness coefficients When the fairness has a value of 1, service types are throttled strictly in proportion to their expected arrival rates for the next control period Thus the control operation is identical to the previous throughput maximisation algorithm When the fairness coefficient is decreased total revenue increases Lower revenue services are rejected more than higher revenue ones As throughput increases in each case, system revenue levels off as the network's throughput is reached for the optimal service mix that is being accepted Note that, for any value of fairness, processor loads are maintained close to the 90% threshold as with the throughput maximisation algorithm

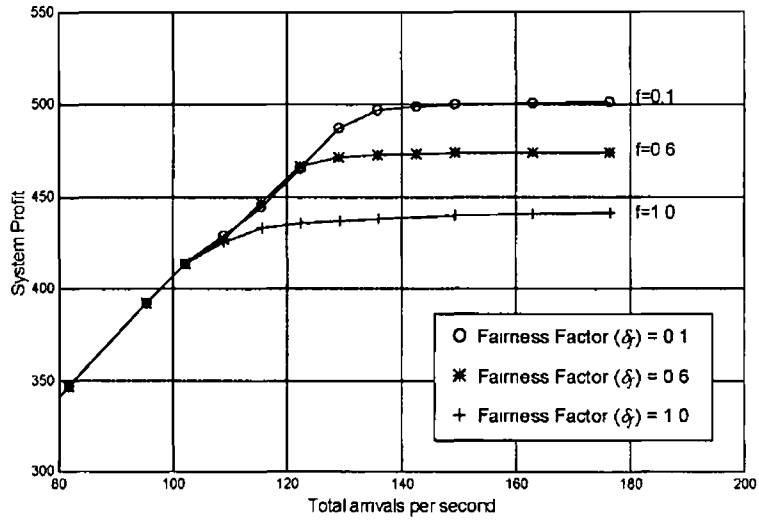


Figure 6 20 Maximum System Revenue Dependence on Fairness Coefficient

Figures 6 21 and 6 22 show the fraction of arrivals accepted for each service for two values of fairness coefficient simulated in Figure 6 20 above The acceptance ratio measure is obtained from the rejection ratio measure given by the simulation and is calculated simply as 1 less the rejection ratio for each service For a low value of fairness (Figure 6 21), low revenue service (Service B) is immediately throttled as increasing arrival intensities cause the system utilisation to approach 90% The two higher revenue services (A and C) are not throttled until the arrival rate has further increased By this stage Service B is approaching complete rejection At high arrival rates the acceptances ratios are decreasing approximately linearly, indicating that the system revenue has saturated Note that although Service C is less than half as profitable as Service A, it is not throttled by very much more than A This is accounted for by the fact that C requires less processing than A and so the revenue per unit processing power is comparable for both services

In Figure 6 22, the fairness constraint has been tightened by specifying a higher value fairness coefficient (0 6) In this case, the low revenue Service B is again throttled first but not as deeply as before The lower revenue service of the remaining two, Service C, is throttled more heavily as a result in order to maintain loading at 90% The high revenue service maintains a high acceptance ratio as, with a fairness coefficient less than 1, the algorithm is still partially driven by revenue maximisation Overall, however, the throttling is fairer to Service B than before

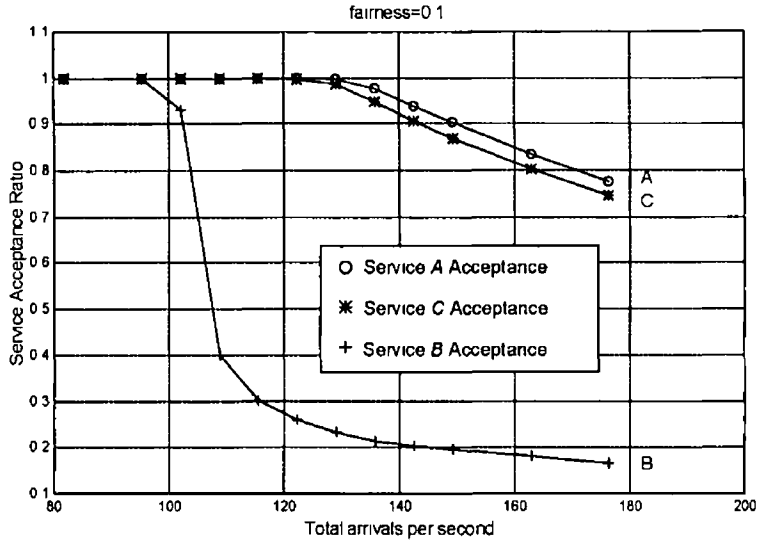


Figure 6 21 Accepted Arrivals for each Service when Fairness is Low

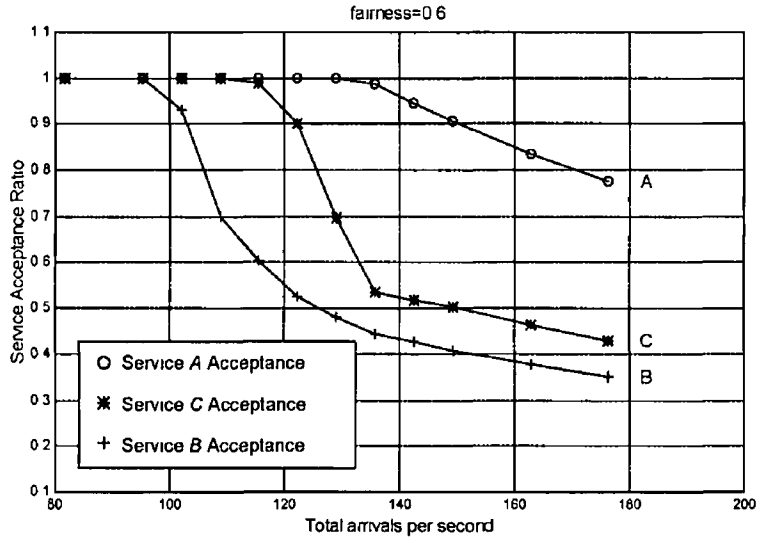


Figure 6 22 Accepted Arrivals for each Service with Moderate Fairness

6 5 1 Note on Two-Phase Revenue Optimal Heuristic

In §5 1 8 1, it was noted that, when arrival rates are low compared to the maximum network throughput, the revenue optimisation problem will not tend to drive all nodes to full capacity and the resulting random splitting may give uneven loading across processors A two-phase approach was proposed where the revenue optimal objective and constraints are used only when throttling is required In the single-phase approach the revenue optimal objective and constraints are always applied We examine the improvement achieved when the two-phase approach is employed

The following low-load scenario is used Arrival rates from all sources are equal with total arrivals to the network of 42 sessions per second The resulting simulated processor loads for the single and two-phase approach are as follows

	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
Processor % Utilisation (Single Phase)	89 2	90 0	46 3	9 60	0 00	0 00	0 00	48 5	17 6	17 3
Processor % Utilisation (Two Phase)	31 9	32 1	31 3	31 5	32 1	32 1	31 7	31 2	32 0	31 1

The two-phase approach has achieved load-balance at approximately 32% loading on all processors, whilst the single-phase approach displays large load imbalances, from 0% to 90% This load imbalance condition of the single-phase approach has the same drawbacks as displayed by the communications cost optimisation and should be avoided

6.6. The Market-based Internal Performance Control Algorithm

A sub-optimal market-based approach to the internal performance control problem was proposed in §5 2 In this case, internal and external controls are effected by means of tokens Token pools are easily mapped to Percentage Thinning coefficients and random splitting probabilities as discussed in §5 2 6 Thus in terms of implementation in the simulator, the methods are the same An auction takes place every 20 seconds, assigns the token pools and maps these to Percentage Thinning coefficients and random splitting probabilities, which are implemented by the simulator in the usual way

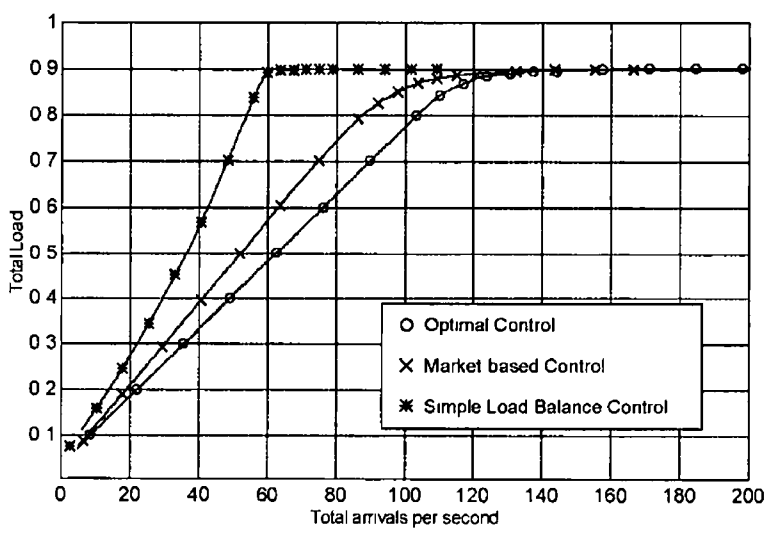


Figure 6 23 Loading for Market Internal and External Controls

We first compare throughput and service session delay to the previous results for the fair optimal algorithm (fairness coefficient is 1) and the simple load balance algorithm, where

relative arrival rates for each service are equal Figure 6 23 gives the load characteristics for the simple load balancing, market and optimal algorithms Again the 90% throttle has not been exceeded and is stable for high arrival rates The market algorithm achieves approximately 82% of the throughput of the optimal algorithm under the equal loading scenario

The average service session delay (Figure 6 24) shows a reasonably good delay characteristic with delays less that twice that of the optimal algorithm for any arrival rate The delay is bounded at a relatively low value for higher throughputs As the market algorithm is throttling the high load Service B more than A or C, the delay is relatively less at high loads compared to the fair optimal algorithm

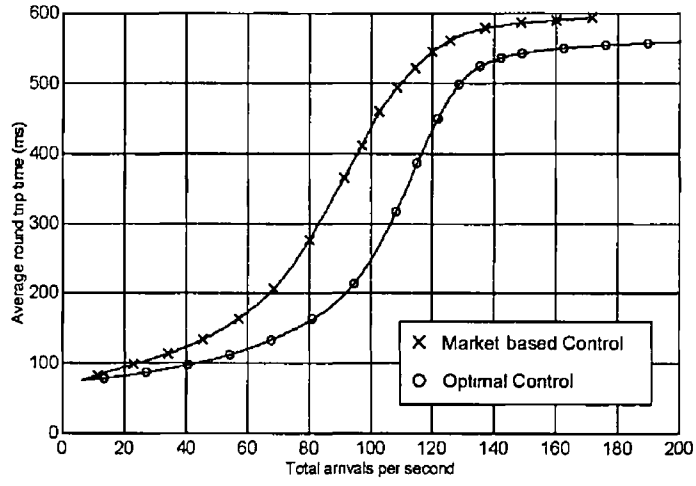


Figure 6 24 Market Service Delays Compared to the Optimum

We compare system revenues, in the high load region, to the optimal algorithm with a fairness coefficient of 1 (Figure 6 25) All relative arrival rates are again equal The market algorithm compares favourably to a fair algorithm in terms of revenue optimisation Of course, the optimal algorithm will perform better when the fairness constraint is relaxed, however the result is still encouraging considering the market-based approach is a simpler heuristic algorithm

The corresponding acceptance ratios for the market algorithm are given in Figure 6 26 and give some insight into algorithm operation Throttling starts much earlier than for the optimal algorithm as the market algorithm is not as efficient and approaches the 90% load threshold earlier The relative throttling rates are similar to the optimal case for a fairness coefficient of 0 6, so the algorithm is behaving reasonably fairly The discrimination between services follows relative values of the service revenue values, as in the optimal case

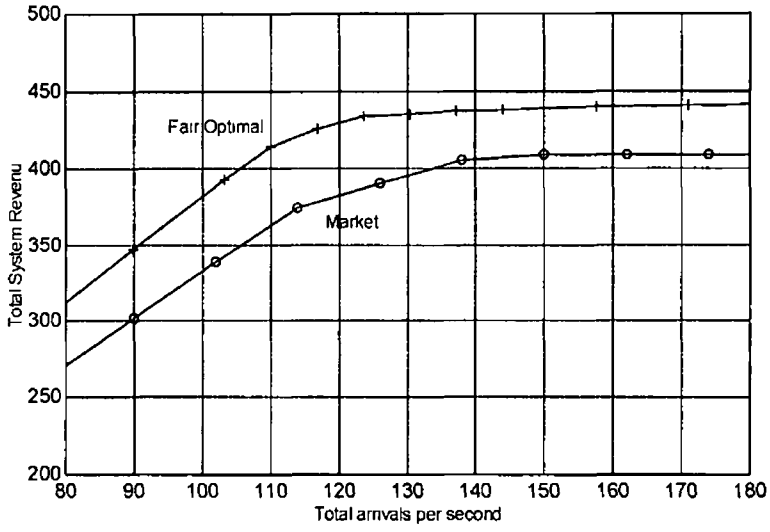


Figure 6 25 Market System Revenue Compared to Fair Optimal when Arrivals are Equal

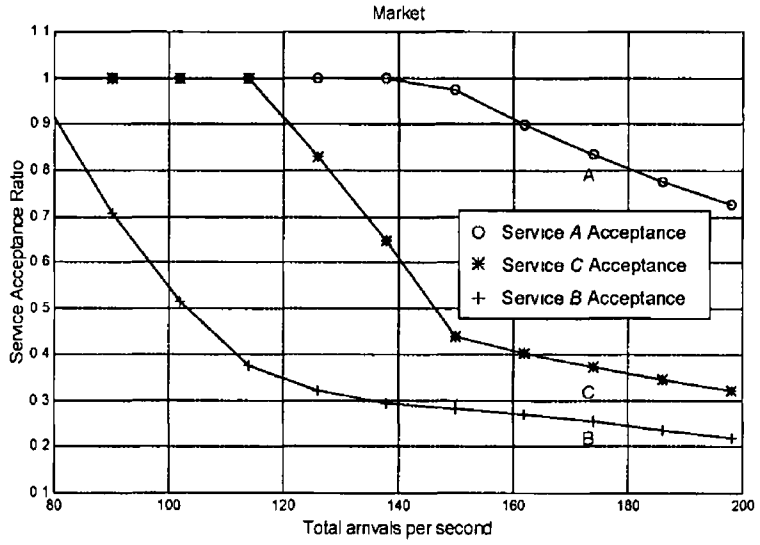


Figure 6 26 Throttling of each Service Type by the Market Algorithm

6.7. Dynamic Controls Under High Load and Varied Service Mix

We have thus far not examined the behaviour of the optimal and market internal and external controls when traffic mix is varied greatly from the design point Preliminary loading and delay results were found for a 1 2 1 traffic mix Initial investigations of the revenue optimal algorithms were conducted at the design point (1 1 1 traffic mix) Here we examine the revenue maximisation algorithms at a wider range of mixes from 1 1 1 to 10 1 1, 1 10 1 and 1 1 10 The market and optimal algorithms are compared in a high-load scenario Arrival rates have been chosen so that the revenue return from the network has saturated Thus we are

examining maximum revenue achievable under different traffic mixes We also look at two values of fairness for the optimal algorithm and compare with the market algorithm

Figure 6 27 shows, total system revenue for different service mixes The service mix is indicated on the x-axis where, for example, at a value of 5 for the curve labelled B, the revenue value indicates total system revenue for a traffic mix of 1 5 1 The arrival intensities in all cases are high enough to saturate the system revenue at the given service mix The set of graphs (A, B and C) in Figure 6 27 are for the optimal algorithm with a fairness coefficient of 1 A similar set, for a fairness coefficient of 0 6, is shown in Figure 6 28

Revenue returns for the market algorithm were found to be relatively invariant, only spanning a range of approximately ±6%, regardless of traffic mix Thus, we simply express the average for the market algorithm in Figure 6 27 and 6 28

From Figure 6 27 it is observed that the effect of changing traffic mix for the optimal algorithm is determined by the relative revenue values for each service In the case of a large proportion of high-revenue Service A arrivals, most revenue is obtained Conversely for a large proportion of low-revenue Service B arrivals, the total revenue decreases The market algorithm can return more revenue in this case, as it is not acting fairly It will throttle Service B arrivals and C arrivals in whatever proportions are necessary to achieve maximum revenue Thus, the market algorithm does not display sensitivity to traffic mix The average revenue over all service mixes for the fair optimal algorithm is approximately 465 profit units, whilst the average for the market algorithm is 391 units

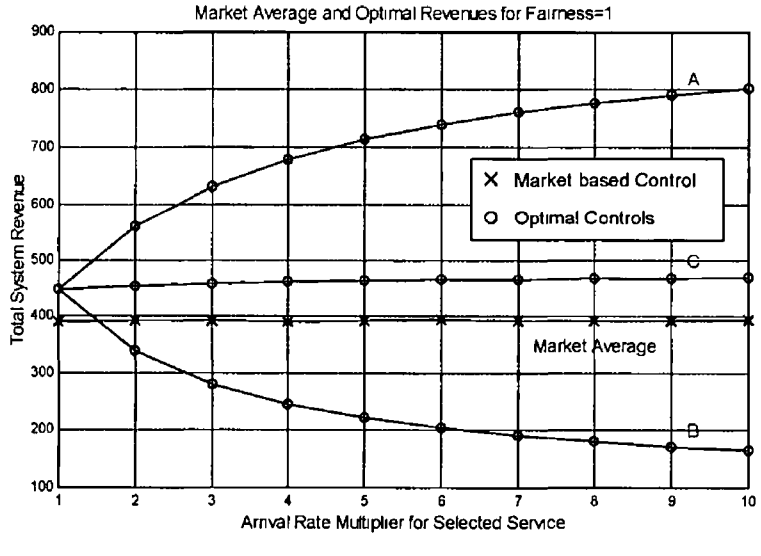


Figure 6 27 System Revenue with Varying Service Mix for High Fairness Revenue-Optimal Algorithm

Figure 6.28 shows the optimal algorithm with the fairness constraint relaxed. A fairness coefficient of 0.6 has been set in this case. The average revenue over all service mixes has increased to 558 profit units and the revenue at any given service mix has increased. The market average is still higher on average than the case where Service B accounts for a high proportion of the traffic.

Note that in each traffic-mix for the optimal algorithm, close to 100% of the available processing capacity on all nodes was utilised. It can be concluded that the dynamic algorithm can function for a wide range of service mixes that are far from the design-point splitting probabilities of the static component placement. For the market algorithm, not all mixes gave 100% utilisation, although the system revenue was maintained constant. However, over the 28 service mixes simulated, only 4 showed 'wastage' of processing power of more than 20%.

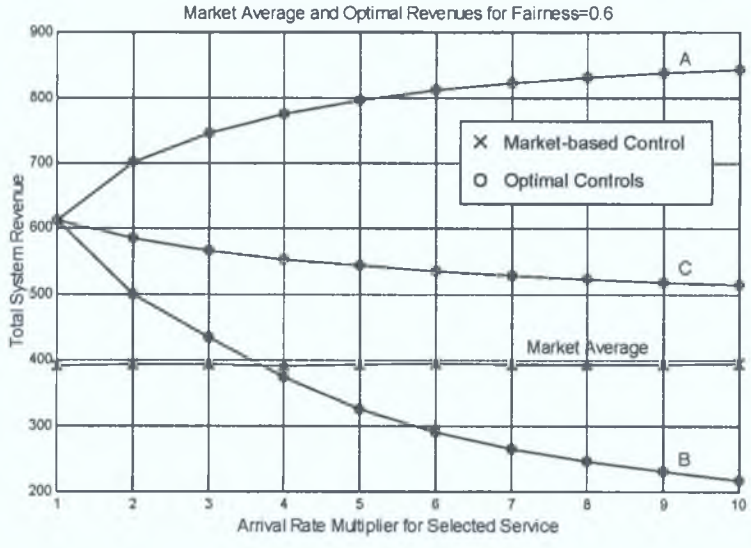


Figure 6.28: System Revenue with Varying Service Mix for Moderate Fairness Revenue-Optimal Algorithm

To make an accurate comparison with the market algorithm, we compare revenue optimality at the same level of fairness. As the fairness coefficient does not relate to the market algorithm, the following independent measure, *fairness index* [Jain *et al.*, 1984], is used.

$$Fairness\ Index = \frac{\left(\sum x_i\right)^2}{n\sum x_i^{\ 2}}$$

where x_i is the ratio of actual throughput to the throughput under fair conditions for each service type i and n is the number of services. The throughput under fair conditions was taken as the total throughput proportioned fairly according to the given service mix. The results are as follows:

The market algorithm has a fairness index of 0 63, averaged over the range of traffic mixes With an appropriately chosen fairness coefficient, the optimal algorithm also has a fairness coefficient of 0 63, averaged over the range of traffic mixes At this level of fairness, the optimal algorithm produces 573 units of revenue, whilst the market revenue is 391 units On this basis, the market algorithm is 68% efficient

6.8. Chapter Summary and Conclusions

In this chapter, the properties of the throughput optimal CO allocation method has been examined and a distribution chosen, giving a reference platform for examination of the dynamic algorithms The allocation method was found to be efficient under different traffic mixes, producing allocations that fully utilised processing power in the network and produced low service time delays The issue of installation cost of CO copies was examined and it was found that cost could be reduced to some degree without limiting throughput Networks with varying numbers of processing nodes were tested and it was found that throughput increased linearly with processing capacity to the point where the gateway nodes became bottlenecks

Also in this chapter, the throughput optimal solution has been compared to a communications cost minimisation approach It was found that there are potential set backs to this approach, as loading on nodes will not necessarily be balanced in the solution This means that there is a tendency for higher and more varied service times in the network and that some nodes are in danger of overload even when the network as a whole is under-loaded As the problem does not have a viable solution for certain values of arrival rates, it would be difficult to implement an admission control with the minimal communications cost approach

The effects of variation in service mix when the random splitting probabilities are fixed demonstrated the need for dynamic controls The performance of the optimal and market dynamic internal and external performance controls was examined and compared to the results for a simple load-balancing scheme The simple load-balancing scheme performs poorly in comparison and illustrates that it is not sufficient to aim only for load balance and to prevent overloads The complexity of the system requires more than a simple greedy approach It can be inferred that other simple algorithms (join-the-shortest-queue and round robin etc), that rely solely on estimates of load levels of processors in the system and have no *a priori* knowledge of message processing times, would also fail to achieve good solutions It is necessary to consider remote communications costs or service times to achieve optimal solutions in distributed environments where communications is resource intensive

The revenue optimal algorithm was examined and it was found to throttle service requests in accordance with the prescribed service revenue values and overall was effective in increasing

network revenues The simple fairness coefficient was found to be useful for limiting disproportionate throttling of low revenue services while still allowing a substantial increase in overall system revenues

The two-phase revenue optimal algorithm was found to have an advantage over the single-phase version The load imbalance condition introduced by the single-phase approach has the same drawbacks as displayed by the communications cost optimisation and should be avoided for that reason

Both the optimal and market algorithms were subjected to a wide range of service mixes at high traffic intensities and found to be stable in terms of overload protection of the network (limiting node utilisation to close to 90%) and in terms of maintaining system revenues at high levels In the case of the optimal algorithm, close to 100% of allowed processing capacity on all nodes was utilised The market algorithm maintained utilisation within 20% below the allowed utilisation When compared at the same level of fairness the market algorithm was found to be approximately 68% efficient in terms of maximising network revenue It was also effective in bounding delays to within approximately twice that of the optimal algorithm

Chapter 7. Conclusions and Future Work

This chapter gives our conclusions in terms of (i) the relation to and the impact on the general area of research, (ii) the properties of the proposed optimal component allocation schemes, (iii) the properties of the dynamic controls proposed and (iv) the properties of the Market-based approach proposed In conclusion, potential topics for future work are identified

7.1. Contributions to the Area of Research

Previously proposed software allocation strategies in the area of distributed systems research have assumed simple models of the underlying system in order that tractable problems may be constructed Methods for optimally minimising delay in networks have been limited to systems to which product-form queuing models apply For more complex systems, where product form assumptions no longer hold, the simpler approach of minimising a generic cost associated with overall communications flow is normally applied Component placement problems, which stipulate a particular type of interaction between communicating entities, are a relatively new addition to the performance control area and have thus far considered a quite small set of performance metrics and constraints There has been little attention to overload protection for distributed systems in the literature with the exception of web server overload protection, which is generally specific to the technologies involved

With regard to dynamic load sharing controls, much of the distributed systems research has focused on scheduling for an independent job model of the workload rather than on load sharing between dependent communicating components Some simple load sharing schemes have been proposed for TINA but dynamic optimal schemes for inter-dependent components have not previously been considered

From the IN literature, it is evident that network-centric approaches, rather than node centric approaches, are gaining in popularity and revenue optimisation has become an important consideration for performance optimisation Overload control is also an important theme and is a prerequisite for most approaches

Considering the state-of-the-art in the area, as outlined above, the optimisation models proposed in this thesis combine the following features giving a more comprehensive and flexible optimisation model for performance control of distributed telecommunication services than those previously proposed:

- A telecom-centric objective of throughput or network revenue maximisation is applied.
- Internal and external controls are integrated to give optimal network-wide load balance and overload protection.
- Multiple service types are considered in the model, as differentiation between service types is an important aspect of telecommunications service provisioning.
- The optimisation model allows more detail of the communications costs to be included. Separate client/server costs are allowed, as opposed to simpler generic communications costs seen in other models.
- Component installation costs may be bounded by limiting the number of component copies in the network.
- Multiple component copies can be modelled in the network.

With regard to fulfilling performance requirement for telecommunications services, the controls have been demonstrated to perform in the following areas:

- *Responsiveness*: Although minimisation of delay was not a direct objective, the controls were demonstrated to bound delays to reasonable levels as a desirable side effect of the throughput maximisation objective.
- *Stability and Reliability*: Overload controls were seen to maintain loads below desired levels and were stable at high offered loads.
- *Profitability and Fairness*: The possibility in trading-off overall network revenue and fairness to customers has been demonstrated.
- *Optimality*: The solutions are optimal in terms of usage of processing capacity in the network and maximisation of revenue.
- *Scalability and Flexibility of Solution*: The solutions to the component placement and internal and external controls were found to utilise all available resources under varying traffic mixes under high loads. The models have been kept general and may be applied to many inter-working scenarios where the network access points are controlled by the operator, for example, gateways from other access technologies (the Web, mobile networks) to a telecommunications service provisioning platform.

7.2. Properties of the Optimal Component Allocation

The following conclusions are made with regard to new work undertaken here in the area of optimal component placement

Existing ideas for optimising software component placement have been found to be useful in the development of more telecom-centric performance approaches for distributed platforms In particular, methods for description of the problems in terms of Linear Programming problems have given an efficient solution method to relatively complex problem constructions Furthermore, with the addition of integer variables to the problems, Mixed Integer Programming has proved flexible for capturing a wide range of competing performance objectives and constraints, which is a required design objective of this work We have found the optimal computational component placement solutions we proposed to have the following properties

- The component placement strategy was found to be robust in terms of allocating resources for multiple services types with different arrival rates into the system
- The throughput of the system was found to scale linearly with processing power to the point where fixed location components introduce bottlenecks Thus analysis with optimisation may be used to identify bottlenecking and to assess the requirements for processing power in the network, given the capacity of the access nodes
- Installation cost limits, associated with multiple component copies, will reduce system throughput if too stringent However, imposing moderate cost limits can reduce the number of component copies without reducing throughput Thus installation costs are useful even in the case where maximum throughput is the main concern
- When the components are relatively fine-grained (as in the case for TINA COs), the optimisation can also produce a good partitioning scheme, by tending towards assigning components to nodes in groups

7.3. Properties of Dynamic Controls

With regard to internal and external dynamic controls for the system an optimal approach has been found to have the following properties and advantages

- The solution to the problem gives both an internal control, through specification of random splitting probabilities, and an external control, via maximum achievable arrival rates Thus, the approach is integrated and all nodes may be protected from overload with the use of throttles only at the gateways The optimality of the approach means that rejection of messages internal to the system, and the associated performance degradation, may be avoided

- Although the static placement of components ultimately determines the maximum performance return, we have found that optimal dynamic random splitting can cope with traffic that varies widely from the design point Thus, limitations of the initial placement can be avoided
- Balancing load without consideration of throughput does not ensure an efficient network Nor can load limits placed on processors alone ensure reasonable delays Throughput maximisation will tend towards balancing load, and will generally give reasonable average delays
- Balancing of load with throughput maximisation means that full utilisation of processing resources under high load may be achieved Although load balance is achieved under high load with communications cost minimisation, the usual objective for component placement strategies, during low average network load some processors may be fully utilised whilst others are almost idle and so there will be no headroom for sudden increases in traffic during the control period
- The profit optimisation approach allowed differential treatment of service types whilst fully maximising utilisation of processing resources in the system The scheme was found to be adaptable to varying mixes of input traffic, retaining the preferred order of customer priorities The fairness parameter employed was effective in allowing trade off between maximum profit and more fair treatment of customers, whilst keeping the problem linear and easily solvable

7.4. Properties of the Market-based Approach

With regard to the market-based approach, the control has been found to perform well in terms of throughput, on average delivering approximately 80% of the throughput of the optimal algorithm It can deliver more consistent revenue returns under varying traffic mixes (although substantially lower on average) than the optimal algorithm, whilst maintaining a degree of fairness in customer treatment The market-based approach has the advantage of allowing non-linear constraints, which can be used to advantage The profit optimal LP problem was found to have a limitation when dealing with low load situations, which leads to uneven loading in the network A correction was required in the form of a two-phase approach This is not necessary with the market algorithm, which allows non-linear constraints Thus the market-based algorithm may be a more natural choice for profit optimisation, although lower overall returns are expected than for the optimal method The market algorithm is also somewhat less complex than the LP and may be more easily implemented and is expected to have shorter execution times

7.5. Future Work

Issues concerning the efficient implementation of complex control algorithms have not been addressed here and this is left for future work. A suggested technique for reducing complexity is to run the algorithms off-line over a range of loading scenarios. The resulting set of solutions may then be used by the controller to choose random splitting and throttling levels appropriate for the current load situation. Note that solutions could also be held at each node, which would then only receive an account of the load situation at the end of control intervals and choose the appropriate control parameters from local information. This scheme could be enhanced with an interpolation algorithm for increased accuracy.

It has been assumed here that the control algorithms have exact knowledge of execution times for message flows in the network. If the estimates are inaccurate, the load control will not perform optimally. A solution may be to adjust the assumed processing times in accordance with variations between load values predicted by the controller and the measured load. A simple control might vary message-processing times linearly to attempt to minimise the error. This could be implemented as a Linear Programming problem. If processing times in the network are assumed to be stochastic, a linear predictive filter may be of use. More complex schemes that do not assume linearity might attempt application of a Neural Network solution. This may be an interesting control problem in its own right as there is complexity in the interaction of the predictive control and the original optimisation problem.

A further area of research might be to consider optimising the initial distribution of objects with respect to the service mix *distributions* during high load rather than simply the average expected traffic volumes for each service. Although the initial CO allocation was found not to be limiting in our case, it may be so depending on the nature of the application and the network. In the suggested scheme, we would presume that the service types have a known joint probability distribution (possibly estimated from traffic studies) describing the likelihood of any particular ratio of traffic from the services occurring. We could then apply stochastic techniques to achieve a component allocation scheme that is optimal in terms of the service mix distribution, rather than simply in terms of the average service mix expected. This approach would be most valuable when the service mix is highly variable. A suggested method for applying this approach is to form a search space for maximising expected throughput. Assessment of each point in the space would require solution to the allocation problem for the corresponding service mix and the resulting solution for maximum throughput would be weighted by the probability of that service mix occurring. An efficient random algorithm, for example a Genetic Algorithm, could be employed to find the maximum.

Thesis Publications

The following articles were produced from the work done in this thesis

[1] C McArdle, R Brennan, N Jones, J Vasic and T Curran, "Implementation Experience with the OMG IN/CORBA Interworking Specification", Proceedings of 6th International Conference on Intelligence in Networks (ICIN 2000), Bordeaux, January 2000

[2] C McArdle, Niklas Widell, C Nyberg, E Lilja, J Nystrom and T Curran, "Load Balancing for a Distributed CORBA-Based SCP", Proceedings of Telecommunications and IT Convergence Towards Service E-volution, 7th International Conference on Intelligence and Services in Networks (IS&N 2000), Athens, Greece, pub Springer, pp 33-48, February 2000

[3] R Brennan, B Jennings, C McArdle and T Curran, "Evolutionary Trends in Intelligent Networks", IEEE Communications Magazine, Volume 38, No 6, pp 86-93, June 2000

[4] R Brennan, B Jennings, C McArdle and T Curran, "Intelligent Networks A Discussion of Current Developments", in Gerald Grant (ed) Managing Telecommunications and Networking Technologies in the 21st Century, pub Idea Group Publishing, pp 1-20, 2001

[5] C McArdle and T Curran, "Optimal Object Placement, Load Distribution and Load Control for Distributed Telecommunication Service Applications", Proceedings of 17th International Teletraffic Congress Teletraffic Engineering in the Internet Era (ITC 17), Salvador, Brazil, pub Elsevier, pp 371-382, December 2001

[6] C McArdle and T Curran, "Performance Controls for Distributed Telecommunications Services", to be submitted to the Journal of Performance Evaluation, pub Elsevier, January 2005

References

[Aguilar & Gelenbe, 1997] J Aguilar, E Gelenbe, "Task Assignment and Transaction Clustering Heuristics for Distributed Systems", Journal of Information Sciences, vol 97, No 1& 2, pp 199-219, 1997

[Anagnostou, 1998] M E Anagnostou, "Optimal Distribution of Service Components", Proceedings of the 5th International Conference on Intelligence in Services and Networks (IS&N98), Antwerp, Belgium, May 1998

[ANSI, 1999] American National Standards Institute (ANSI), "Intelligent Networks", 1999

[Arvidsson *et al*, 1997] A Arvidsson, S Pettersson, L Angelin, "Profit Optimal Congestion Control in Intelligent Networks", Proceedings of the 15th International Teletraffic Congress, pp 911-920, pub Elsevier Science B V, 1999

[Asensio *et al*, 1998] J I Asensio, V A Villagra, J I Moreno, J Berrocal, "An Approach to Electronic Brokerage in TINA Environment", Proceedings of the 5th International Conference on Intelligence in Services and Networks, Belgium, pp 339-350, 1998

[Aspval *et al*, 1980] B Aspval, R E Stone, "Khachiyan's Linear Programming Algorithm", Journal of Algorithms, vol 1, no 1, pp 1-13, 1980

[Avramopoulos & Anagnostou, 2002] I C Avramopoulos, M E Anagnostou, "Optimal Component Configuration and Component Routing", IEEE Transactions On Mobile Computing, Vol 1, No 4, October - December 2002

[Banks, 1998] J Banks (ed), "Handbook of Simulation Principles, Methodology, Advances, Applications and Practice", pub John Wiley & Sons Inc, 1998

[Baskett *et al*, 1975] F Basket, K M Chandy, R R Muntz and F G Palacios, "Open, Closed and Mixed Networks of Queues with Different Classes of Customers", Journal of the ACM, vol 22, pp 248-260, 1975

[Bastarrica *et al*, 1998] M C Bastarrica, A A Shvartsman, S A Demurjian Sr, "A Binary Integer Programming Model for Optimal Object Distribution", Technical Report CSE-TR-98-1, Department of Computer Science and Engineering, University of Connecticut, April 1998

[Baumgartner & Wah, 1990] K M Baumgartner, B W Wah, "Computer Scheduling Algorithms Past, Present, and Future", First Workshop on Parallel Processing, National Tsing Hua University, Taiwan, December 1990

[Billionnet *et al*, 1992] A Billionet, M C Costa, A Sutter, "An Efficient Algorithm for a Task Allocation Problem", Journal of the Association of Computing Machinery, vol 29, No 3, pp 502-518, July 1992

[Blair *et al*, 2001] G S Blair, G Coulson, M Clarke, N Parlavantzas, "Performance and Integrity in the OpenORB Reflective Middleware", Reflection, 2001, pp 268-269, 2001

[Borst, 1995] S C Borst, "Optimal Probabilistic Allocation of Customer Types to Servers", Proceedings of 1995 ACM SIGMETRICS joint International Conference on Measurement and Modeling of Computer Systems, Ottawa, Ontario, Canada, pp 116-125, 1995

[Bowen *et al*, 1992] N S Bowen, C N Nikolaou, A Ghafoor, "On the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems", IEEE Trans Computers, Vol 41, No 3, pp 197-203, March 1992

[Capellmann & Pageot, 1999] C Capellmann, J M Pageaot, "A TINA Service Platform Integrated with Current Intelligent Network Systems", Proceedings of TINA'99, Turtle Bay, Hawaii, pp 295-301, April 1999

[Cardellini *et al*, 1999] V Cardellini, M Colajanni, P S Yu, "Dynamic Load Balancing on Web-Server Systems", IEEE Internet Computing, Vol 3, No 3, pp 28-39, 1999

[Casevant & Kuhl, 1998] T Casevant, J Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", IEEE Transactions of Software Engineering, Vol 14, pp 141-154, February 1988

[Charnes, 1953] A Charnes, W W Cooper, A Henderson, "An Introduction to Linear Programming", pub John Wiley and Sons Inc , 1953

[Chen & Mohapatra, 2003] H Chen, P Mohapatra, "Overload Control in QoS-aware Web Servers", Computer Networks The International Journal of Computers and Telecommunications Networking, Vol 42, Issue 1, pp 119-133, May 2003

[Choo *et al*, 2002] C L Choo, M Kihl, L S Wei, I Chai, "Performance and Load Issues in TINA Networks", TINA Workshop, Kuala Lumpur, Malaysia, 2002

[Chu *et al*, 1980] W W Chu, L J Holloway, M T Lan, K Efe, "Task allocation in distributed data processing", Computer, Vol 13, Issue 11,pp 57-69, November, 1980

[Clearwater, 1996] S H Clearwater (ed), "Market-based Control A paradigm for Distributed Resource Allocation', pub World Scientific Publishing Co , 1996

[Dantzig, 1953] G B Dantzig, "Computational Algorithm of the Revised Simplex Method", Report RM 1266, The Rand Corporation, Santa Monica, California, 1953

[Davidsson *et al*, 2000] P Davidsson, B Carlsson, S Johansson and M Ohlin, "Using Mobile Agents for IN Load Control", Proceedings of 2000 IEEE Intelligent Network Workshop (IN'2000), F J Scholtz (ed), Cape Town, South Africa, May 2000

[Denning & Buzen, 1978] P J Denning, J P Buzen, "The Operational Analysis of Queueing Network Model Models", ACM Computing Surveys, vol 10, pp 225-262, 1978

[Diestel, 1997] R Diestel, "Graph Theory", Graduate Text in Mathematics, Volume 173, Second Edition, Springer-Verlag, New York, 1997

[Efe, 1982] K Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", IEEE Transaction on Computers, pp 50-56, June 1982

[Elsadek & Wells, 1999] A A Elsadek, B E Wells, "A Heuristic Model for Task Allocation in Heterogeneous Distributed Computing Systems", The International Journal of Computers and their Applications, Vol 6, No 1, March 1999

[ETSI, 1994] European Telecommunications Standards Institute (ETSI), "Intelligent Network (CS1) Core INAP Protocol Specification", ETS 300 374-1, ed 1, September 1994

[Eurescom, 1997] EURESCOM, "Introduction to Distributed Computing Middleware in Intelligent Networks A Eurescom P508 Perspective", OMG DTC Document orbos/97-09-11, September 1997

[Franks & Woodside, 1998] R G Franks, C M Woodside, "Performance of Multi-level Client-Server Systems with Parallel Service Operations", Proceedings of First International Workshop on Software and Performance (WOSP98), pp 120-130, Santa Fe, October 1998

[Franks, 1999] R G Franks, "Performance Analysis of distributed server Systems", PhD Thesis, Carleton University, Ontario, Canada, 1999

[Gelenbe, 1999] E Gelenbe, G Pujolie, "Introduction to Queueing Networks", 2nd edition, pub John Wiley & Sons Inc , 1999

[Hac & Gao, 1998] A Hac, L Gao, "Analysis of Congestion Control Mechanisms in an Intelligent Network", International Journal of Network Management, vol 8, pp 18-41, 1998

[Harju, 1994] J Harju, T Karttunen, O Martikainen, "Intelligent Networks", Chapmen & Hall, 1994

[Herzog & Megdanz, 1997] U Herzog, T Magendanz, "From IN toward TINA - Potential Migration Steps", Proceedings of 4th International Conference on Intelligence in Services and Networks, pub Springer, Cernobbio, Italy, May, 1997

[Hunt & Scott, 1999] G C Hunt, M L Scott, "The Coign Automatic Distributed Partitioning System", Proceedings of Third Symposium on Operating System Design and Implementation (OSDI), 1999

[ITU-T, 1993] ITU-T, Rec Q 721-Q 766, "Specifications of Signalling System No 7", March 1993

[ITU-T, 1997] ITU-T, Rec Q1221, "Introduction to Intelligent Network Capability Set 2", September, 1997

[ITU-T, 1993a] ITU-T, Rec Q 1201, "Principles of Intelligent Network Architecture Recommendation", 1993

[Iyer *et al.*, 2000] R. Iyer, V. Tewari, K. Kant, "Overload Control Mechanisms for Web Servers", Performance and QoS of Next Generation Networks, Nagoya, Japan, pp. 225-244, Nov 2000.

[Jackson, 1963] J. R. Jackson, "Jobshop like Queueing Systems", Management Science, vol. 10, pp. 131-142, 1963.

[Jacobson & Lazowska, 1982] P. A. Jacobson, E. D. Lazowska, "Analysing Queueing Networks with Simultaneous Resource Possession", Communications of the ACM, vol. 25, No. 2, pp. 142-151, February 1982.

[Jain, 1991] R. Jain, "The Art of Computer Systems Performance Analysis", pub. John Wiley & Sons Inc., 1991.

[Jain *et al.*, 1984] R. Jain, W. Hawe, D. Chiu, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," DEC-TR-301, September 26, 1984.

[Jennings *et al.*, 1999] B. Jennings, R. Brennan, R. Gustavsson, R. Feldt, J. V. Pitt, K. Prouskas and J. Quantz, "FIPA-compliant Agents for Real-time Control of Intelligent Network Traffic," Computer Networks (The International Journal of Computer and Telecommunications Networking), Vol. 31, No. 19, pp. 2017-2036, August 1999.

[Jennings, 2001] B. Jennings, "Network-Oriented Local Control for SS.7/IN", PhD Thesis, School of Electronic Engineering, Dublin City University, 2001.

[Karmarkar, 1984] N. Karmarkar, "A New Polynomial-time Algorithm for Linear Programming", Combinatorica, Vol. 4, pp. 373-395, 1984.

[Kazuo, 2001] T. Kazuo, K. Kenzo, "Temperature Parallel Simulated Annealing and its Applications", IPSG SIG Notes, Mathematical Modelling and Problem Solving, Abstract No.011-001, 2001.

[Kernighan & Lin, 1973] B. Kernighan and S. Lin, "An Effective Heuristic Procedure for Partitioning Graphs", The Bell System Technical Journal, pp. 291-308, Feb 1970.

[Kfil & Ahmad, 1998] M. Kafil, I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems", IEEE July-Sept 1998.

[Kihl & Nyberg, 1997] M. Kihl, C. Nyberg, "Investigation of Overload Control Algorithms for SCPs in the Intelligent Network", IEEE Proceedings, Vol. 144, No. 6, pp. 419-424, December 1997.

[Kihl *et al.*, 1999] M. Kihl, N. Widell, C. Nyberg, "Load Balancing Algorithms for TINA Networks", Proceedings of the 16th International Teletraffic Congress, Vol 3b, pp. 999-1008, Edinburgh, Scotland, 1999.

[Kihl *et al.*, 1997] M. Kihl, C. Nyberg, H. Warne, P. Wollinger, "Performance Simulation of a TINA Network", Proceedings of IEEE Globecom'97, Phoenix, USA, 1997.

[Kihl, 1998] M Kihl, "On Overload Control in a TINA Network", Telecommunications 1998, Conference Publication No 451, IEEE, 1998

[Kleinrock, 1975] L Kleinrock, "Queueing Systems, Volume1 Theory", pub Wiley-Interscience, 1975

[Lagarias & Todd, 1990] J Lagarias, M Todd, "Probabilistic Analysis of the Simplex-Method", Contemporary Mathematics, vol 114, pp 21-34, 1990

[Lazowska *et al*, 1984] E D Lazowska, J Zahorjan, G S Graham, K C Sevcik, "Quantitative System Performance Computer System Analysis Using Queueing Network Models", pub Prentice-Hall, 1984

[Lee & Bic, 1989] C Lee, L Bic, "On the Mapping Problem using Simulated Annealing", Computers and Communications, Proceedings of 8th Annual International Phoenix Conference, 1989

[Lewis *et al*, 2005] M Lewis, B Alidaee, G Kochenberger, "To Model and Solve the Uncapacitated Task Allocation Problem", Operations Research Letters, Vol 33, Issue 2, pp 176-182, March 2005

[Lo, 1988] V M Lo, "Heuristic Algorithm for Task Assignment in Distributed Systems", IEEE Transaction on Computers, Vol 37, No 11, pp 1384-1397, November 1988

[Lodge, 2000] F Lodge, "An Investigation into Intelligent Network Congestion Control Techniques", PhD Thesis, Dublin City University, Ireland, 2000

[Mampaey, 2000] M Mampaey, "TINA for Services and Advanced Signaling and Control in Next-Generation Networks", IEEE Communications Magazine, October 2000

[McArdle *et al*, 2000] C McArdle, R Brennan, N Jones, J Vasic, T Curran, "Implementation Experience with the OMG/CORBA Interworking Specification", Proceedings of 6th International Conference on Intelligence in Networks (ICIN 2000), Bordeaux, January 2000

[Nilson, 1971] N J Nilson, "Problem Solving Methods in Artificial Intelligence", McGraw - Hill, New York, 1971

[OMG, 1995] Object Management Group, "Object Management Architechture Guide", Revision 3 0, 1995

[OMG, 1996] Object Management Group, "Intelligent Networking with CORBA", OMG DTC Document telecom/96-12-02, December 1996

[OMG, 1997] Object Management Group, "White Paper on CORBA as an Enabling Factor for Migration from IN to TINA A P508 Perspective", OMG DTC Document telecom/97-01-01, January 1997

[OMG, 1999] Object Management Group, "IN/CORBA Interworking", OMG document /dtc/99-12-02, December 1999

[OPNET, 2004] Opnet Modeler Home Page, www opnet com, last visited June 2004

[OSL, 2004] IBM OSL Home Page, www research ibm.com/osl/, last visited June 2004

[Papadimitriou & Steglitz, 1982] C Papadimitriou, K Steglitz, "Combinatorial Optimization", Prentice Hall, Englewood Cliffs, New Jersey, 1982

[Papoulis, 1984] A Papoulis, "Probability, Random Variables, and Stochastic Processes", 2nd edition, pub McGraw-Hill, 1984

[Parhar & Rumsewicz, 1995] A Parhar, M P Rumsewicz, "Critcal Congestion Issues in the Evolution of Common Channel Signalling Networks", Proceedings of the 14th International Teletraffic Congress, Juan-Les-Pins, France, 1994

[Park, 1997] K Park, "A heuristic approach to task assignment optimization in distributed systems", Systems, Man, and Cybernetics, 1997 "Computational Cybernetics and Simulation", IEEE International Conference, 1997

[Patel *et al*, 2000] A Patel, K Prouskas, J Barria, J Pitt, "A Computational Economy for IN Load Control using a Multi-Agent System", Journal of Network and Systems Management, vol 8, No 3, September 2000

[Pham & Betts, 1992] X H Pham, R Betts, "Congestion Control for Intelligent Networks", Proceedings of 1992 International Zurich Seminar on Digital Communications, Intelligent Networks and their Applications, Zurich, 1992

[Pinto *et al*, 1999] A S Pinto, E J Olivera, L F Faina, E Caradozo, "TINA - based Environment for Mobile Multimedia Services", TINA International Conference, Hawaii, pp 54-65, 1999

[Purao *et al*, 2002] S Purao, H K Jain, D L Nazareth, "ODE a tool for distributing object-oriented applications", Information and Management, Vol 39, No 8, pp 689-703, September 2002

[Ramakrishnan *et al*, 1993] S Ramakrishnan, L Dunning, T Nitsch, "An Integrated Optimal Task Assignment Policy", ACM-SAC, 1993

[Rolia & Sevcik, 1995] J Rolia, K Sevcik, "The Method of Layers", IEEE Transactions on Software Engineering, Vol 21, No 8, pp 689-700, August 1995

[Ross & Yao, 1991] K W Ross, D D Yao, "Optimal Load Balancing and Scheduling in a Distributed Computer System", Journal of the ACM, Vol 38, No 3, April 1985

[Schmidt, 1997] D C Schmidt, "Distributed Object Computing", IEEE Communications Magazine, February 1997

[Schrijver 1986] A Schrijver, "Theory of Linear and Integer Programming", pub Wiley-Interscience, 1986

[Senar *et al.*, 2003] M. A. Senar, A. Ripoll, A. Cortes, E. Luque, "Clustering and Reassignment-based Mapping Strategy for Message-Passing Architectures", Journal of Systems Architecture: the EUROMICRO Journal Archive, Vol. 48, Issue 8-10, pp. 267-283, March 2003.

[Shen & Tsai, 1985] C. C. Shen, W. H. Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems using a Minimax Criterion", IEEE Transactions on Computers, vol. C-34, No. 3, March 1985.

[Shousha *et al.*, 1998] C. Shousha, D. Petriou, A. Jalnapurkar, K. Ngo, "Applying Performance Modelling to a Telecommunication System", WOSP98, Santa Fe, N.M., 1998.

[Silaghi & Keleher, 2001] Silaghi, Keleher, "Object Distribution with Local Information", Proceedings of 21st International Conference on Distributable Computing Systems, Mesa A.Z., U.S.A., April 2001.

[Singh & Youssef, 1996] H. Singh, A. Youssef, "Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithms", Proceedings of 5th IEEE Heterogeneous Computing Workshop (HCW 96), 1996.

[Smith, 1995] D. E. Smith, "Ensuring Robust Call Throughput and Fairness for SCP Overload Controls", IEEE/ACM Transactions on Networking, Vol. 3, No. 5, 1995.

[Sperryn *et al.*, 2000] J. Sperryn, S. Mohapi, R. Achterberg, H. E. Hanrahan, "Performance Issues and Load Monitoring in the TINA DPE", Proceedings of TINA 2000 Conference, Paris, p 13, September 2000.

[Stone *et al.*,1997] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", IEEE Transactions on Software Engineering. SE-3,1, pp. 85-93, January 1997.

[Talbi & Muntean, 1991] E. G. Talbi, T. Muntean, "A New Approach for the Mapping Problem: A Parallel Genetic Algorithm", 2nd Symposium on High Performance Computing, Montpellier, North Holland, pp. 71-82, Sep. 1991.

[Tantawi & Towsley, 1985] A. N. Tantawi, D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems", Journal of the Association for Computing Machinery, Vol.32, No. 2, pp. 445-465, April 1985.

[TINA-C, 1997] TINA-C, "TINA Service Architecture", Version 5.0, 16 June, 1997.

[TINA-RET, 1999] TINA-C, "Retailer Reference Point (Ret)", Version 1.1, Jan 1999.

[Trigila *et al.*, 1998] S. Trigila, F. U. Bordoni, K. Raatikainen, B. Wind, P. Reynolds, "Mobility in Long-Term Architectures and Distributed Platforms", IEEE Personal Communications, pp. 44-55, August 1998.

[Vanderbei, 2001] R. J. Vanderbei, "Linear Programming: Foundations and Extensions", 2nd edition, pub. Kluwer Academic Publishers, 2001.

[Walsh, 1985] G. R. Walsh, "An Introduction to Linear Programming", 2nd edition, pub. Wiley-Interscience, 1985.

[Wang & Sevcik, 2000] H Wang, K C Sevcik, "Experiments with Improved Approximate Mean Value Analysis Algorithms", Performance Evaluation, Vol 39, No 1-4, pp 189-206, February 2000

[Wang & Morris, 1985] Y T Wang, R J T Morris, "Load Sharing in Distributed Systems", IEEE Trans On Computers, Vol C-34, No 3, pp 204-217, March 1985

[Widell *et al*, 1999] N Widell, M Kihl, C Nyberg, "Measuring Real-time Performance in Distributed Object Oriented Systems", Proceedings of Performance and Control of Network Systems III, SPIE Photonic East '99, Boston, USA, September 1999

[Wolf & Yu, 2001] J L Wolf, P S Yu, "On Balancing the Load in a Clustered Web Farm", ACM Transactions on Internet Technology (TOIT) Archive, Vol 1, Issue 2, pp 231-261, November 2001

[Woodside *et al*, 1995] C M Woodside, J E Neilson, D C Petriu, S Majumdar, "The Stochastic Rendez-Vous Network Model for Performance of Synchronous Client-Server-like Distributed Software", IEEE Transactions on Computers, Vol 44, No 1, pp 20-34, January 1995

[Woodside, 1996] C M Woodside, "Performability Modelling for Multi-Layered Service Systems", Proceedings Third Internation Workshop on Performability of Computer and Communications Systems, Bloomingdale, Illinois, USA, September 7th-8th, 1996

[X/OPEN, 1995] X/OPEN, "Inter-Domain Management Specifications Specification Translation", X/Open Preliminary Specifications, Draft August 9, 1995

[Yu *et al*, 1986] P S Yu, S Balsamo, Y H Lee, "Dynamic Load Sharing in Distribution Database Systems", IEEE Proceedings of 1986 Fall Joint Computer Conference, pp 675-683, 1986