DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING

An Investigation into Intelligent Network Congestion Control Strategies

Fiona Lodge, B.Eng.

PhD Thesis, February 2000

Under Supervision of Dr. Thomas Curran & Dr. Dmitri Botvich

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

Signed: _____ ID No.: _____

Date: _____

Acknowledgements

I have a lot of people to thank for the quality of this thesis. During its lifetime, it has passed through many stages of evolution, this process being driven by the comments, observations and criticisms of Dmitri Botvich, Adrian Newcombe, Brendan Jennings, Tommy Curran and Declan Gavin. Thanks guys!

Dmitri and Adrian also deserve further honourable mention for contributions above and beyond the call of duty. Their detailed grilling of my ideas and strategies led to more refinements than I care to remember. If I could, I'd have their names on the cover too.

And finally, I would like to dedicate this thesis to the memory of my grandparents, Johnny Kelly and Gretta Lodge. They would have burst with pride if they had lived to see it, but would almost definitely never have gotten around to reading it!

Table of Contents

Table of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 6

Abstract

This thesis examines the congestion control issues that arise in Intelligent Networks, when it is necessary to support multiple service types with different load requirements and priorities. The area of Intelligent Network (IN) congestion control has been under investigation for over a decade, but in general, the models used in this research were over-simplified and all service types were assumed to have the same priority levels and load requirements at the various IN physical elements. However, as the IN is a dynamic network that must process many different service types that have radically different call load profiles and are based on different service level agreements and charging schemes, the validity of the above assumptions is questionable. The aim of this work, therefore, is to remove a number of the classic assumptions made in IN congestion control research, by:

- developing a detailed model of an IN, catering for multiple traffic types,
- using this model to establish the shortcomings of classic congestion control strategies,
- devising a new IN congestion control strategy and verifying its superiority on the model.

To achieve these aims, an IN model (both simulation and analytic) is developed to reflect the physical and functional architecture of the network and model the information flows required between network entities in order to execute services. The effectiveness of various classic active and reactive congestion control strategies are then investigated using this model and it is established that none of these strategies are capable of protecting both the Service Control Point and Service Switching Points under all possible traffic mixes and loads. This is partially due to the fact that all of these strategies are based on the use of fixed parameters (and are therefore not flexible enough to deal with IN traffic) and partially because none of these strategies take into account the different load requirements of the different service types.

A new, flexible strategy is then devised to facilitate global IN congestion control and cater for service types with different characteristics. This strategy maximises IN performance by protecting all network elements from overload while maximising network revenue and preserving fairness between service types during overload. A number of factors determining the relative importance or *weight* of different traffic types are also identified and used by the strategy to maintain call importance during overload. The efficiency of this strategy is demonstrated by comparing its operation to that of the best classic IN overload controls and also to a new strategy, which has scalable and dynamic behaviour (and which was devised for the purpose of providing a fair comparison to the optimisation strategy). The optimisation-based strategy and dynamic strategy are found to be equally effective and far superior to the classic strategies. However, the optimisation algorithm also preserves relative importance and fairness, while maximising network revenue - but at the cost of a not insignificant processing overhead.

Chapter 1

Introduction

This thesis examines the area of Intelligent Network (IN) congestion and proposes strategies and algorithms to protect an IN from this phenomenon. In this chapter, the terms *congestion* and *IN* are defined and a brief description of the main issues involved with IN congestion control is provided. Then, the primary objectives of our research in this area are presented, followed by an outline of the structure of this thesis.

1.1 General Background

In the area of telecommunications, the term *congestion* refers to the scenario where the amount of work offered to a telecommunications network element becomes so great that the element is no longer capable of dealing with it. For example, if a Public Switched Telephony Network (PSTN) switch is suddenly flooded with call requests (requests from users to establish a voice connection or execute a service), but is not fast enough (does not have sufficient processing power) to deal with these requests at the rate at which they are arriving, then the requests build up in the input buffer. A further contributing factor is that the processing of a call by the switch is distributed across the lifetime of the call and it is vital, in order to maintain efficiency, that once processing has begun on a call, it completes successfully. Therefore all calls that are initially accepted by the switch require more processing time at a later stage. This leads to even more demand being placed on the processors and reduces further the rate at which the switch can deal with newly arriving call requests. At this point the switch is *overloaded* or *congested*, the result of which is that users experience unacceptably long delays in receiving dial tone, during call setup (Post Dialling Delay) and at various other points in the call. Further, many calls are not established successfully and the operation of the switch may be placed in jeopardy - in the worst case, the switch can break down completely. It is therefore vital that a strategy for reducing the load on the switch processors (i.e. the amount of work they have to do), namely a *congestion* (or *overload*) *control strategy*, is implemented at the switch to prevent serious overload and to ensure that the switch is well protected and operating at optimum efficiency at all times.

The area of congestion and congestion control techniques has been well researched and documented, with many methods existing, each with its own advantages, disadvantages and therefore, applications. These control techniques are usually examined and compared using simulations and analytic models of simple single-processor systems to represent network elements. These models assume that a network comprises multiple switches, with each switch being responsible for protecting itself from overload, either by refusing all call requests outright (referred to as *blocking)*, by refusing a proportion of requests (referred to as *throttling*) or by notifying other switches of the overload situation and telling them to route call requests over alternative paths. This type of model is therefore sufficient for representing the operation of a PSTN or an Integrated

Services Digital Network (ISDN), where the network is comprised of a number of physical entities, each carrying out similar functions and none more important to the operation of the whole than any of the others.

The Intelligent Network does not have this kind of architecture. It was standardised by ITU-T [ITU_IN] in order to specify a network architecture that allows users to create and maintain services quickly and easily and which provides all authorised users in the network with the ability to access any of the offered services. This objective is met by defining a number of IN elements (called Physical Entities (PEs)) that together allow a service to be accessed and executed. Service Switching Points (SSPs) receive service requests from users and are responsible for all switching functions associated with terminating the call and also for the invocation of service logic in a Service Control Point (SCP) – a PE of the IN at which service logic resides (and therefore effectively the 'core' of the IN). SCPs control the execution of all services by transmitting requests to and receiving information from the SSPs, Service Data Points (SDPs, which provides SCPs with service- and user- specific data maintained in network databases) and Intelligent Peripherals (IPs, which provides the functionality for the exchange of information with the user). Therefore, in order to execute even the simplest service for a user, several messages must be transmitted between an SCP and other IN PEs over a Signalling System Number 7 (SS7) network during the course of only one call. This, of necessity, results in very large quantities of traffic to and from the SCP. Therefore, the SCP is the principle bottleneck in the IN architecture, and the PE most likely to become overloaded. It is also, however, the most important PE in the network (as it contains all service logic programs) and it is therefore crucial that it be protected all times. Due to both the distributed nature of the IN and the fact that the SCP is the most important element in the network, the modelling of network behaviour cannot be accomplished accurately using the simple single-processor model that is generally applied to PSTN and ISDN. Therefore, to investigate IN congestion control issues, a new, more appropriate model must be developed.

The objective of an efficient congestion control strategy for the IN (or indeed any congestion control strategy) is to successfully complete as many service requests for as many users as possible, while keeping response delays as low as possible. However, the added complexity of the IN architecture means that a number of other issues must be taken into consideration, among which are:

- SCPs are the most important PEs in the IN, as all IN service requests must receive processing at least once at an SCP. Therefore, if an SCP becomes congested, the number of services that will be able to execute will be severely limited and it must therefore be protected at all costs. On the other hand, it should process as many calls as is safely possible at all times, even during congestion (i.e. it should not expend much of its capacity on the rejection of call requests).

- Multiple messages may arrive at an SCP during a single execution of a service at that SCP- i.e. new call requests, once accepted, usually (depending on the type of service) lead to the later arrival of requests associated with the same call. These requests may not be blocked, but must be processed further in order to optimise SCP throughput (i.e. the amount of SCP capacity spent processing calls which terminate successfully). For example, a simple service may involve the arrival of two requests at an SCP (e.g. for a number translation service, SCP arrivals would consist of the initial service request from the SSP followed by another as a reply to a data lookup request from the SDP), while a relatively complex service would generate many more SCP arrivals, require more processing and be therefore more likely to cause or exacerbate a congestion situation.
- In the future, it is likely that SSPs will handle non-IN traffic, as well as IN service requests. This will have implications on IN performance, because indiscriminate blocking of calls by the SSP in order to protect itself may result in severe under-utilisation of the SCP. Also, non-IN calls have lower processing overhead (by a factor of approximately 2.5) than IN requests, so a policy of throttling all calls equally, without distinguishing between them, may prove inefficient or even ineffective.
- The provision of IN services to customers tends to be based on Service Level Agreements between the service provider and customer. As such, different services will have different levels of importance, agreed arrival rates and tariffs (as well as different processing requirements at the SCP). This implies that there may be a need for a priority-based congestion control system.

1.2 Research Objectives & Methods

The principle objective of this work is to develop a new IN congestion control strategy which performs better than the IN strategies which have been used in industry over the last decade. To ensure the superiority of the new strategy, it should be compared with the best of these existing strategies in an IN network model which simulates closely (with minimal assumptions) the structure of a real Intelligent Network, the behaviour of its various functional components and the trends and variations which tend to occur in IN traffic.

The first step in reaching this goal is therefore to develop a comprehensive model of the Intelligent Network and to use it to examine the behaviour of the most commonly used SCP congestion control strategies in industry today (SSP overload is generally not considered to be a factor in IN congestion control and therefore SSP protection tends not to be included in the implemented strategies). The results of this investigation then highlight the problems that exist in these strategies. This provides insight into the characteristics that would be desirable in a new IN

congestion control strategy and therefore allows the requirements on this new strategy to be more explicitly stated and refined.

The second step is to enhance the IN model in order to examine the effects on IN performance of mixed IN and non-IN traffic at a finite-capacity SSP (i.e. an SSP which may experience congestion). This investigation proves that the behaviour of an SSP congestion control strategy may significantly affect the overall performance of the IN and, in doing so, demonstrates the advantages of combining SCP and SSP overload controls into a single global IN overload control strategy.

The third step consists of deriving a global IN congestion control strategy that protects both SCP and SSPs, while exhibiting all desirable characteristics identified during the first step described above. Inclusion of this new strategy in the IN model is then followed by rigorous testing and comparison with other strategies in order to verify its superior effectiveness and efficiency.

1.3 Thesis Outline

Chapter 2 of this thesis provides all necessary information about the state of the art in the areas relevant to this work. This includes a brief description of the architecture and operation of the IN, followed by a summary of the results and conclusions of investigations into congestion control in general and IN congestion control in particular. Chapter 3 provides the background information required to aid understanding of the simulation and analytic modelling techniques and congestion control algorithms that will be presented in following chapters.

Chapter 4 investigates methods for the protection of the SCP and documents the comprehensive multi-processor Intelligent Network model that was developed to facilitate this investigation. The effectiveness of classic congestion control detection methods (such as Queue Length Control, Call Count Control and Load Measure Control) for SCP protection is compared, as is the operation of the Call Gapping and Percent Thinning throttle algorithms. Then, the Window congestion control strategy is compared with the best of the above detection methods combined with the best throttle. Finally, a number of points are raised regarding areas in which these strategies' performance is undesirable and how they should be enhanced to improve their effectiveness.

Chapter 5 first presents a new version of the IN model used in Chapter 4 which was enhanced based on the observations made in the conclusions of that chapter. The effect of the presence of non-IN calls at finite capacity SSPs on IN congestion control algorithms is evaluated and it is concluded that a strategy which dynamically manages congestion control jointly at the SCP and SSPs provides better performance than a strategy in which all elements are protected

independently. Then, just such a global strategy (which is based on the mathematical optimisation of revenue) is presented and results of its behaviour under various types and levels of overload are given.

Then, in Chapter 6, to establish whether this revenue optimisation-based strategy provides the best possible IN performance for any load condition, its operation is compared with those classic strategies that were found to have the best performance in Chapter 4. Also, a new version of a classic strategy (in which a dynamic Call Count Control detection method is combined with a hybrid Percent Thinning/Call Gapping throttle) is derived, to ensure a fair comparison with the optimisation-based strategy.

Chapter 7 then summarises the main conclusions of the work and provides a number of recommendations regarding issues that should be addressed and algorithms that should be used when developing a congestion control strategy for use in an Intelligent Network.

Chapter 2

State of the Art

In this chapter, the background information about the Intelligent Network architecture and standards is summarised, as are the results of the research which has taken place in the IN congestion control arena to date. Section 2.1 describes the Intelligent Network in terms of justification for its development, its standard evolution path and the IN Capability Set 1 architecture. Section 2.2 provides an introduction to the general concepts of congestion control and Section 2.3 describes the research that has taken place in IN congestion control over the last ten years.

2.1 The Intelligent Network (IN)

2.1.1 Justification for IN Development

Up until the mid 1980's, services offered by network operators to users consisted principally of basic call connectivity. Since then, however, service technology has grown greatly, with the requirement for more complex services constantly on the increase. The structure of the Public Switched Telephony Network (PSTN) as it stands is not very compatible with the need to supply services, as to do so requires service logic to be available on all PSTN switches. The Integrated Services Digital Network (ISDN) standard [ITU_ISDN] includes the specification of a number of services (e.g. Abbreviated Dialling, Call Forwarding, Call Transfer etc.), but again, for an ISDN switch to have the capability to offer ISDN services, the code for the services must reside locally to the switch. Some of the difficulties associated with making services available in both PSTN and ISDN include:

- The principle problem with the process of service provisioning is that, in PSTN and ISDN, services are localised - in other words, if the software for a service is loaded at a network node, only users directly attached to that node may use that service. Therefore:
  - in order to be able to allow a customer to avail of a service, the appropriate functionality must be loaded at their local node.
  - Also, if a service is to be altered or upgraded, the code must be changed at every node offering the service.
  - This makes the provisioning and maintenance of a service both very difficult and very slow.
  - It is also very wasteful of resources in that the same service software is replicated at a wide range of locations.
- Service developers have the added problem of vendor dependence, in that software and hardware differs greatly between switches provided by different switch vendors and many different brands and types of switches may be available in one network. It is therefore difficult

to ensure that any developed software will work correctly on any given switch, and different code must be written for different switches. Most switch vendors supply some basic services already installed on their switches, but these do not necessarily function correctly when interacting with other switches made by another vendor.

- Service creation is an expensive process, both in terms of time and money, as there is no standardised environment available for the development of services. Some switch vendors have developed proprietary service creation environments that increase the speed of the development process for those vendors, but as they are not standardised, correct service operation across switches provided by different vendors cannot be guaranteed.
- Different software versions across switches add further complications both in the development and execution of services.

As the role of services in networks increased in importance, the necessity arose for a network architecture which addressed the above problems and allowed network operators and service providers to design, implement and maintain services as efficiently as possible to maximise the possible income. Requirements on this service-friendly network included:

- A reduction in the length of the service design and development phase, by providing a standard development environment comprising a set of reusable function blocks and tools to facilitate the rapid design of a service.
- Much shorter deployment and provisioning phases, achieved through centralising all service execution software so that the logic for a new service would need to be installed at relatively few locations in the network in order to make it available for all customers on the network. This centralisation would also simplify the task of upgrading and maintaining services.
- Independence from switch hardware and software vendors - switch vendors would be required, in order to remain competitive, to provide a standard set of functions in their switches to ensure compatibility with the overall network operation and the services residing in it.
- The ability for all users to avail of a service, no matter where they are located in the network.

The Intelligent Network was developed in order to meets these needs - i.e. to facilitate the creation and operation of services within a telecommunications network. The concepts of the IN have been under design since 1988 and are currently widely in use in the USA, Australia, Japan and Europe. Some of the most popular services used in today's telecommunications market, including call manipulation services (such as Call Forwarding, Call Transfer, Call Waiting), Freephone, Televoting, Credit Card Calling and Premium Rate services, are offered via IN.

2.1.2 Evolution of the Intelligent Network

The International Telecommunications Union (ITU) have adopted a 'Chinese Box' approach in the development of IN standards in that each version of the standards (called *Capability Sets*) is a superset of its predecessor, as shown in Figure 2.1 below.

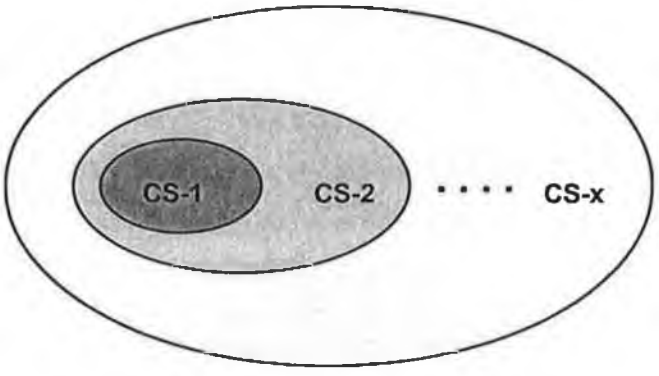


Fig. 2.1: Sequencing of Capability Sets

The first version of the IN standards, ITU Capability Set 1, was released in 1993. CS-1 defines the fundamental architecture of the IN in terms of the IN Conceptual Model (INCM). This INCM addresses only the basic operation of the network in terms of the actions at and the interactions between IN Physical Entities (PEs) required in order to execute a service. CS-1 was the first in-depth standard detailing the concept of the IN and is responsible only for describing the basic principles of service execution and network architecture. Therefore, CS-1 has a number of limitations, in terms of how useful it is to service providers in the development of the types of services that are desired today. These limitations include:

- No network management or service creation features were investigated - no specifications were produced for the IN Service Management Function (SMF) or Service Creation Environment Function (SCEF).
- In IN CS-1, services can only be implemented for Type A calls - i.e. calls which have, at all times, only one originating and one terminating party. Therefore, CS-1 cannot facilitate the creation of Call Transfer, Call Forwarding, Call Waiting, Conference Calling, or any other service that involves more than two call parties at a time.

The specification of Capability Set 2 standards was very slow – the standard did not become available until the end of 1997 – i.e. four years after CS-1. This standards development was not, however, fast enough for service providers, who needed to be able to create and provide a wider range of services than could be encompassed by CS-1. Therefore, IN equipment vendors and IN service providers have developed their own proprietary versions of IN, which are based on the CS-1 INCM, but which provide added functionality to facilitate the offering of Type B (multi-party)

services – an example of this is the Ericsson SCP, which is based on the use of their proprietary CS1+ [EricssonCS1+].

CS-2 expanded on existing CS-1 concepts and added many new concepts in areas not addressed by CS-1 [Q1221]. Among these are:

- A wider range of functionality was specified to allow the development of narrowband Type B services, such as audio-conference calling. This included both the specification of new Service Independent Building Blocks (SIBs) and the extension of the Basic Call State Models (BCSMs) and INAP (Intelligent Network Application Part – the application layer of the SS7 protocol stack).
- High level guidelines were provided for supporting service management services and service creation. This included the specification of Service Management Service Features and interfaces to management, based on TMN (Telecommunication Management Networks) principles [ITU_TMN], interfaces (i.e. the X interface) and protocols (i.e. the Common Management Information Protocol (CMIP)).
- The issue of interworking between INs and other networks was addressed in CS-2. The IN to IN interworking (i.e. interworking across the boundaries of different IN domains) included the specification of mechanisms for interworking between Service Control Points (SCPs), Service Data Points (SDPs) and Service Management Points across IN domain boundaries. An Intelligent Access Function (IAF) was specified to provide access to an SCF in an IN from a non-IN structured network. The security issues that will arise at the boundaries between network domains have also been addressed by CS-2. These interworking facilities will enable IN standards to meet the open market demands for Open Network Provisioning (ONP), i.e. they will permit service providers with service software located in one IN to make it available (as third party service providers) to customers in other networks. These specification will also, even within a single IN domain, improve the operation of the network by providing redundancy and backup systems - for example, multiple SCPs could supply the same service, so that if a problem arose at one SCP, requests could be re-routed to other SCPs supporting the same services. Note, however, that CS-2 still requires a single point of control – i.e. at any one time during execution of a service, an SSP should never have to interact with more than one SCP.
- Some support has been provided for multimedia-type services and for services involving either personal or terminal mobility. This support is now available at the logical level (i.e. SIBs have been defined to support various features of these types of services), but this support is not yet reflected at protocol level.

In parallel with the CS-2 standardisation effort at ITU, the Telecommunications Intelligent Networking Architecture (TINA) consortium was also founded to investigate how principles of IN and TMN could be applied to the specification of an architecture for broadband services [TINA97]. The TINA specifications are based on the use of CORBA (Common Object Request Broker Architecture [CORBA99]) for service logic specification and ATM (Asynchronous Transfer Mode [ATM99]) for broadband communication, and are currently quite influential in contribution to the definition of standards in the Object Management Group (OMG) (e.g. for access to CORBA-based services) and, to a lesser extent, in the ITU (e.g. the standardisation of ITU-ODL in Study Group 10). It may therefore be predicted that TINA and TMN will all be very influential in the specification of IN CS-3 (due out later in 2000), which is therefore highly likely to encompass:

- Full IN/TMN integration, including full technical specification of the IN-SMF and SCEF,
- Full IN/ATM integration, including functionality (and protocol support) to offer broadband multimedia services, such as video conferences, joint document editing services and auctioning services,
- Full support for personal/terminal mobility services.

Also, as the telecoms and Information Technology (IT) domains continue to converge, it is likely that various other standards bodies which have been established to advance computing technologies, e.g. the OMG, which standardises CORBA for distributed software processing and the IETF (Internet Engineering Task Force), which specifies Internet standards (e.g. Internet Protocol (IP) version 6), will also be influential in the specification of CS-3, but the impact of this work on the IN CS-3 standards is less clear. For example, it is unclear whether the SS7 (Signalling System No. 7) will remain as the underlying protocol stack in CS-3. There is a possibility that CORBA's IIOP (Internet Inter-Orb Protocol) running over IP will become a candidate for this role. Also, the functionality for offering distributed service logic (i.e. where the logic of services will no longer reside at a single physical element (the IN SCP)) and the facility for expressing functionality in terms of OMG's Interface Description Language (IDL) may prove very useful in IN specification and may therefore be incorporated in CS-3.

However, at the moment, most real implementations of IN are still based on CS-1 and, as a result, it is with networks of this type that congestion is currently an issue. Therefore, the model that was developed and described in this report was developed to meet CS-1 specifications and all congestion control research carried out was based on the physical architecture of IN CS-1. This is not very restrictive, as all results and solutions remain valid in CS-2 and only require extending to include issues related to IN-IN interworking (this issue is already being examined by e.g. [Kawamura96] and [Swensen96]). The CS-1 results may also prove relevant to CS-3 networks, if

the standard is based on the use of SS7 (in conjunction with a gateway between the SS7 TCAP (Transaction Capabilities Application Part) layer and CORBA) for transport of control messages. Of course, performance management in a distributed programming domain such as CORBA is a completely separate issue, but it is perceived as being outside the scope of this thesis.

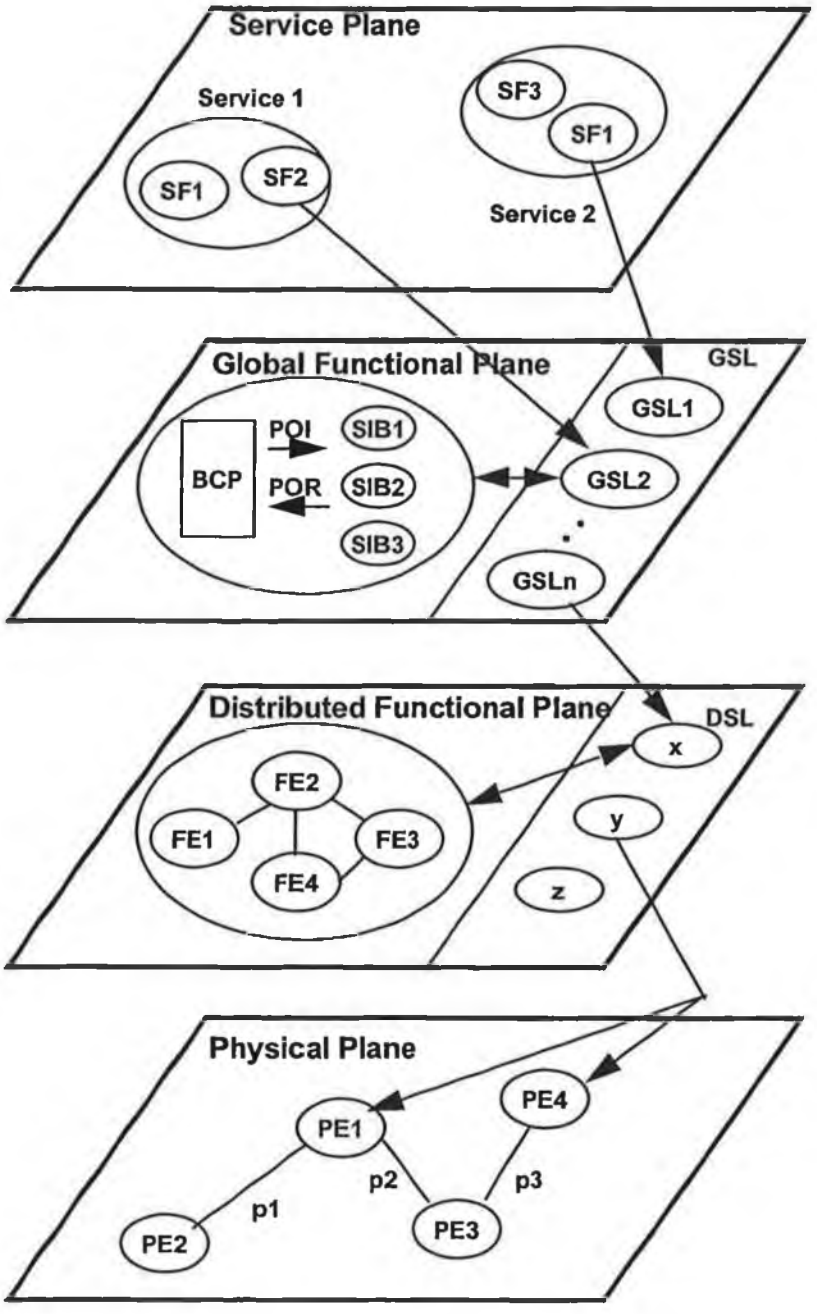
We, therefore, will now explain the functionality of the IN as defined by the CS-1 standards, as being the basic network architecture on which all congestion control research has been carried out in this thesis.

2.1.3 The Architecture and Operation of IN CS-1

The simplest way of explaining the concept of the IN is by introducing the IN Conceptual Model. This provides a planar view of the implementation of a service within the IN. There are four planes in the model, the highest plane representing services as discrete units, with each lower level further examining the operation and execution of the services. At the lowest level, on the Physical Plane, a full breakdown of the actions, interactions and information flows required to execute a service are represented. The conceptual model is shown in Figure 2.2 below.

2.1.3.1 The Service Plane (SP)

This is the highest plane in the Conceptual Model and is described in [Q1202]. At this level, services are described only in terms of how they behave. Services may be distinguished as marketable products that are made up of one or more *Service Features* assembled together. A Service Feature offers a limited amount of functionality to the user and may also be a service in its own right. An example of a service feature that is also classified as a service would be the Abbreviated Dialling Service. The Virtual Private Network (VPN) Service would be a prime example of a service consisting of multiple service features, as a VPN may offer, among others, the Abbreviated Dialling, Call Transfer and Call Forwarding features.



Acronyms:

SF	Service Feature	GSL	Global Service Logic
BCP	Basic Call Process	POI	Point of Initiation
POR	Point of Return	SIB	Service Independent Building Block
DSL	Distributed Service Logic	FE	Functional Entity
PE	Physical Entity	px	Protocol x

Fig. 2.2: The CS-1 Intelligent Network Conceptual Model

2.1.3.2 The Global Functional Plane (GFP)

The GFP describes the functionality of the IN on an abstract level [Q1213] in that it does not deal with how functionality is realised or where it is located within the network. At this level, a service or service feature is perceived as consisting of discrete blocks, called *Service Independent Building Blocks (SIBs)*. As this layer is independent of network structure, SIBs do not really exist as distinct

entities at any location in the IN, but merely represent functionality within the network that is needed to carry out tasks in order to provide the required service. Thirteen SIBs have been defined in CS-1, among which are:

- Verify - this SIB is used to check the format of any input strings.
- Charge - this defines when special charging features or rates are to be applied, including, for example, reverse charging and premium rate charging.
- Queue - this SIB allows calls to be queued at a destination. Announcements may be read to waiting customers until it is their turn to be served.
- User Interaction - this SIB represents any interactions between the network and the user, including the reading of announcements and the collection of digits.
- Translate - this SIB uses input data to reference information. In other words, it represents a database lookup action.
- Screen - this compares an identifier against a list. It could be used as a security measure, to represent the comparison of a Personal Identification Number (PIN) keyed in by the user against a list of authorised users.
- Basic Call Process (BCP) - this is a special SIB which represents basic call functionality and processing. It is therefore responsible for recognising when a service has been requested and when a branch to other SIBs is required.
- When designing or representing a service using SIBs, the relevant SIBs are linked together in chains and are invoked by the BCP via a *Point of Initiation* (POI). The BCP supplies the SIBs with the any *Call Instance Data* (CID - information specific to one call request, including for example, the calling line identifier, the PIN keyed in by the user etc.) required to process the call and execute the service. When a chain of SIBs has completed execution, control is returned to the BCP via a *Point of Return* (POR).

In order to demonstrate how SIBs may be used to represent a service, a GFP representation of a Freephone service will be provided. The behaviour of this Freephone service may be defined as follows: the service is activated when a user picks up their phone and dials a ten digit number, beginning with the string "1800". This number is associated with a particular service subscriber, and an attempt is made to establish a connection between the user and the subscriber. If the call terminates successfully, the subscriber is billed for the call. A global functional plane representation of the setup of this service is shown in Figure 2.3.

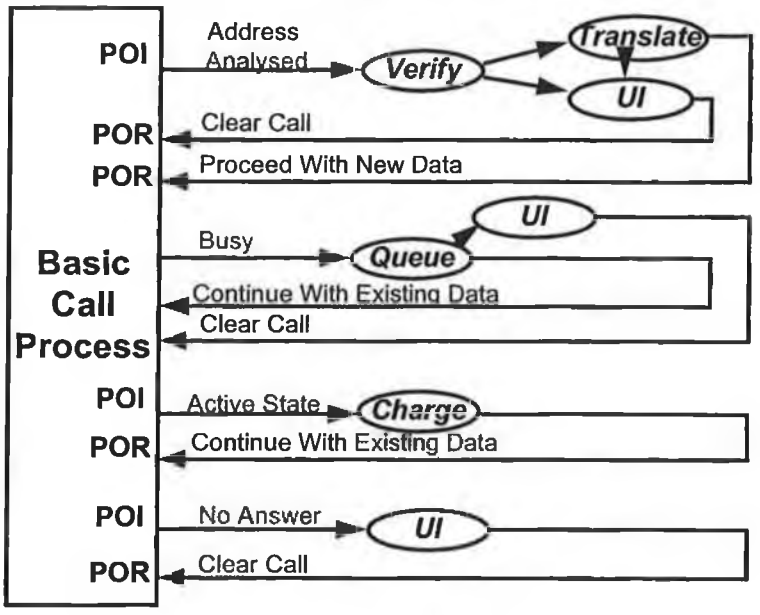


Fig. 2.3: GFP Representation of the Freephone Service Setup

The operation of this model is as follows: the BCP signifies that the analysis of the digits entered by the user and the recognition of a request for the freephone service has taken place by branching at the *Address Analysed* POI to the relevant SIB chain. The first SIB, Verify, establishes that the user has correctly entered a ten digit number. If the number has been entered incorrectly, the User Interaction (UI) SIB is invoked to read an announcement (for example, "Please try again, ensuring that you enter ten digits. Thank you.") to the user and control is returned to the BCP via a *Clear Call* POR. The BCP is then responsible for terminating the call. If the number has been entered correctly, the Translate SIB uses the dialled number to reference the actual Destination Number (DN) of the subscriber. The DN is then returned to the BCP via a *Proceed With New Data* POR, and call processing continues with routing to this new DN. When call setup is being attempted, there are three options as to how the call may progress:

- If the subscriber's line is engaged, a *Busy* POI leads to a Queue SIB, which holds the call in a queue until the line becomes available. Any problem will lead to an announcement being read and the call being cleared, but if everything remains in order, control will be released via a *Continue with Existing Data* POR to the BCP once the line becomes free.
- If the call request is accepted, an *Active State* POI leads to the Charge SIB, which specifies that the terminating party is to be billed for the call.
- If the call is not answered, it is possible to access the User Interaction SIB via a *No Answer* POI and cause an announcement to be read to the user before clearing the call.

2.1.3.3 The Distributed Functional Plane (DFP)

The DFP describes how the various elements of functionality are distributed across the network [Q1214]. The operation of the network is explained in terms of Functional Entities (FEs), each of which carry out specific FE Actions (FEAs) and communicate with each other through Information Flows (IFs). To retain openness, the physical location of the FEs is not addressed in the DFP. Figure 2.4 depicts all FEs defined within CS-1.

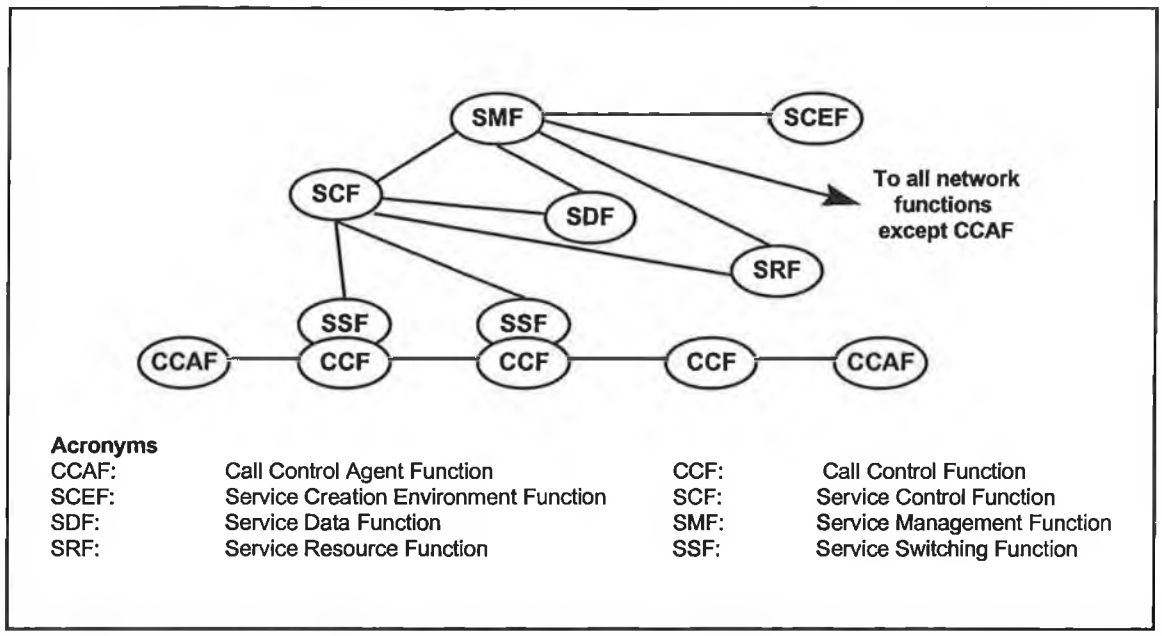


Fig. 2.4: Functional Entities in the IN CS-1 Distributed Functional Plane

The Service Creation Environment Function (SCEF) allows service providers to develop Service Logic Programs (SLPs) quickly and easily. The SCEF is a subfunction of the Service Management Function (SMF) that has overall responsibility for the deployment, provisioning and maintenance of services and for the upkeep of data on the network. Note that no behaviour is specified for either the SMF or SCEF in IN CS-1.

- The Service Control Function (SCF) is responsible for controlling the execution of services. This task requires that the SCF manages the execution of SLPs and is also responsible for handling the transmission of messages to the Service Data Function, the Service Resource Function and the Service Switching Function and the interpretation of results from these functions.
- The Service Data Function (SDF) interprets SCF-generated requests, accesses (reads or writes) data in the database and sends results back to the SCF.
- The Service Resource Function (SRF) provides the functionality for interactions between the network and users i.e. the reading of announcements, collection of digits typed in by the user etc.

- The Service Switching Function (SSF) acts as an interface between the SCF and the Call Control Function (CCF). It interprets messages from the SCF and translates them into instructions for the CCF and also builds information from the CCF into INAP messages for transmission to the SCF.
- It is within the CCF that call processing is handled through the maintenance and manipulation of a Basic Call State Model (BCSM - this models all the possible states a call can be in, along with the requirements needed to pass from one state to another and all possible routes between states. Two BCSMs have been specified in CS-1. The Originating BCSM (O-BCSM) models the states of the originating call (e.g. Onhook, Call Authorisation, Number Analysis, Call Routing etc.), while the Terminating BCSM shows all possible states in which the terminating call may find itself. Note that it is the CS-1 BCSMs which primarily restrict the use of IN CS-1 to the provision of Type A calls.). The CCF is therefore responsible for the detection of service requests at any stage in a call and for notifying the SCF accordingly, by passing the request to the SSF. It also alters the state of a call according to instructions provided by the SCF via the SSF. The CCF is also connected to the Call Control Agent Function (CCAF).
- The CCAF provides connectivity between the CCF and the customer.

Each FE within the network is capable of carrying out a number of actions (FEAs). This is done through the execution of blocks of code within the FE. A set of Information Flows is defined within CS-1 as a message set for passing information between FEs. The SIBs from the GFP may be realised in the DFP as a series of FEAs and IFs. For example, the Translate SIB uses input data as a key to obtain output information. On the DFP, this consists of a FEA within the SCF to build a request with the input data as key, an information flow to the SDF, a FEA in the SDF to look up the requested information and code it into a message and finally, an information flow back to the SCF. This realisation can be extended to services. Taking again the freephone representation on the GFP and making the assumption that the call proceeds without any hitches, a simplified DFP realisation of how this service would cause a call to be setup is shown in Figure 2.5. Note that, in the diagram, the SSF and CCF are modelled together - this is a common method of representing their operation, as in reality, they are very closely linked. So, in showing the decomposition of the service into FEAs and IFs, three functional entities are represented - the SSF/CCF, the SCF and the SDF - r3 and r6 are different communications media between the functions.

Operation of the freephone service is user-driven - when the customer goes offhook an Originating BCSM is created to monitor the progress of the call. After the freephone number has been dialled, it is analysed by the CCF and the '1800' string at the start of the number is recognised as being a request for service. The SSF builds an *Analysed Information* IF containing the dialled digits and sends it to the SCF. The SCF creates an instance of the freephone SLP. This instance invokes FEA 9111, which builds a *Query* IF containing the freephone number as information key and sends it to

the SDF. FEA 4111 in the SDF uses the key from the *Query* IF (i.e. the freephone number) to reference the database to find the actual DN of the freephone subscriber. The DN is found and encrypted in a *Query Result* IF as the outcome of the search. FEA 9112 in the SCF then takes this outcome and builds it into a *Select Route* IF as the destination routing address. This IF is then passed to the SSF, where it is interpreted as a command to continue call processing by selecting a route to the newly supplied DN. The issue of charging is not shown here but would be quite simple to implement - at the point where the call is successfully established, an IF is sent to the SCF which (in some manner not addressed in CS-1) informs the billing system that the subscriber is to be billed for the call.

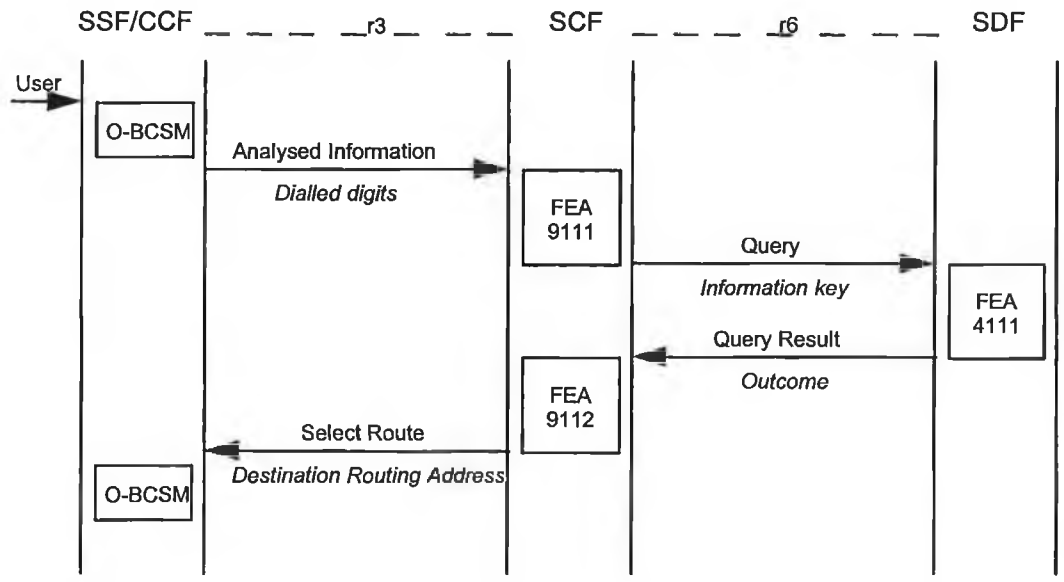


Fig. 2.5: Freephone Service Decomposition

2.1.3.4 The Physical Plane (PP)

On the physical plane, details are provided in [Q1215] as to the physical aspects of the IN. The Physical Entities (PEs) that form the intelligent network, the FEs realised within them and the protocols by which they communicate are described at this level. The IN consists of the following PEs:

- The Service Switching Point (SSP) - user access to service functionality is provided through the SSP, which handles call processing, detects service requests and provides connectivity to the SCP and other SSPs in the network. The SSP contains three discrete functions - the CCAF, the CCF and the SSF.
- The Service Control Point (SCP) - the Service Control Function (SCF) resides here along with the SLPs whose execution it manages.
- The Service Data Point (SDP) - this houses the SDF and is connected directly to the SCP. It contains all network data relevant to the execution of services.

- The Intelligent Peripheral (IP) - the SSP maintains a number of channels between itself and the IP, which contains the Service Resource Function (SRF). Interactions occur between the SRF and users when the SSP opens a channel between them. The IP receives instructions relating to announcements and digit collection directly from the SCF and, when necessary, returns any acquired information.

All communications between SSPs and SCPs and between SCPs and IPs occur over an SS7 network using the TCAP part of the protocol (see [Q1218]). The PEs and their interconnections, along with the FEs realised within them, are shown in Figure 2.6 below.

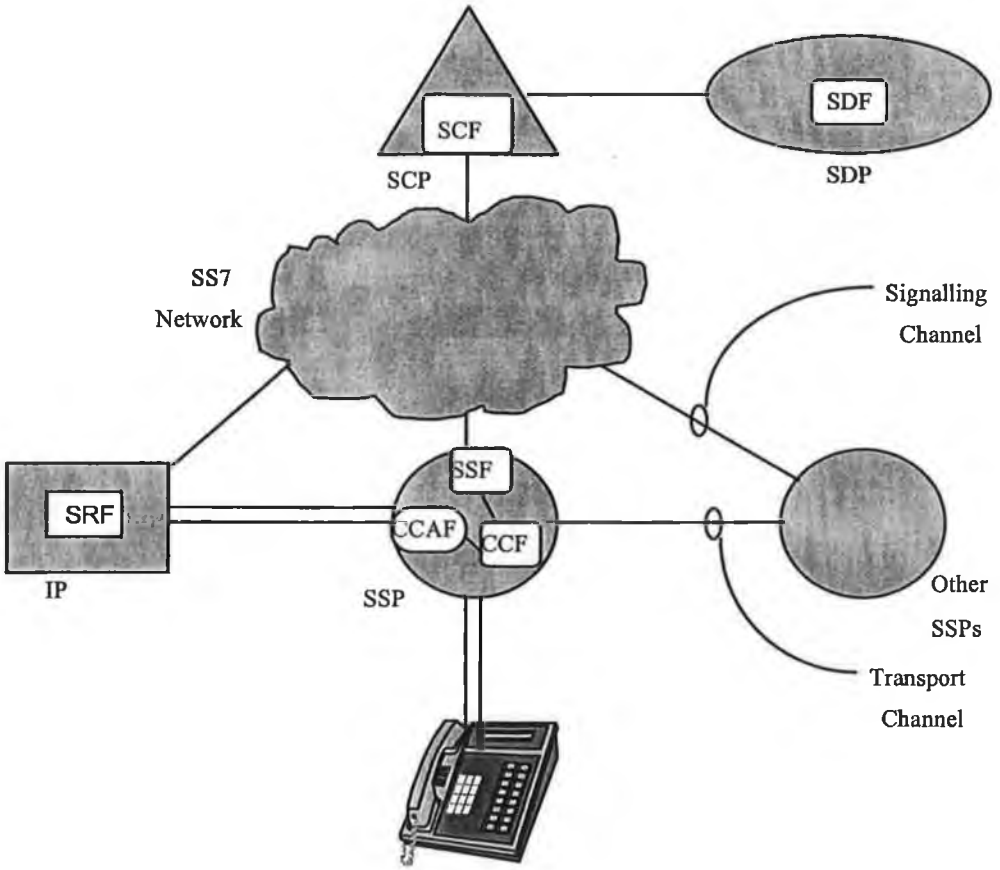


Fig. 2.6: Physical Architecture of the CS-1 Intelligent Network

2.2 Congestion Control

Congestion, or overload, of any system may be defined as that case when the arrival rate of requests at the system exceeds the service rate of the system. Therefore, the number of calls to be dealt with increases continuously as the input buffers build up. If the system has no method of decreasing the number of waiting requests, all processing time is spent trying to deal with the backlog. The result is that, as the input queue grows, each new request must wait longer and longer to gain access to the processor, until the lengths of the delays to users become unreasonable, with the result that users begin to abandon their call attempts and retry, thus increasing the input queue further. This could eventually lead to the even worse scenario of a complete overload situation, resulting in the system being unable to process the presented load and malfunctioning. These delays, abandonments and malfunctions result in fewer calls being handled by the system. In all cases, whether the system to be protected is a telecoms switch, a telecoms network, a LAN server or a data network, this result is highly undesirable. Take, for example, a telecoms switch. From the users point of view, an overload at their local switch (or a remote switch through which they are trying to communicate) means that they are either presented with long delays in call handling or are unable to complete a call successfully - this entails a serious drop in the quality of service to network users. From a network operator's perspective, not only does congestion result in losses of revenue, but may also, in today's open market, lead to the loss of unsatisfied customers. It is therefore vital for both network operators and subscribers that overload does not occur, and so congestion control is one of the top priorities in the design and operation of telephone switches.

2.2.1 Basic Requirements on Congestion Control Strategies

In order to describe and compare various congestion control strategies, it is necessary to initially define the requirements for a successful control strategy. Each strategy will then be evaluated to judge its efficiency at meeting each requirement - each strategy will have its strengths and weaknesses. An analysis and comparison of the performances of each technique will then allow the overall best method to be selected. The requirements were summarised by [Korner91] and may be listed as follows:

1. It must be impossible for a complete breakdown to take place due to the input queue being overwhelmed.

2. The system must retain good throughput characteristics at all times, i.e. the amount of resource capacity spent accepting calls and completing processing of accepted calls should not decrease

during an overload. The characteristics should approach the ideal shown in Figure 2.7 below as much as possible.

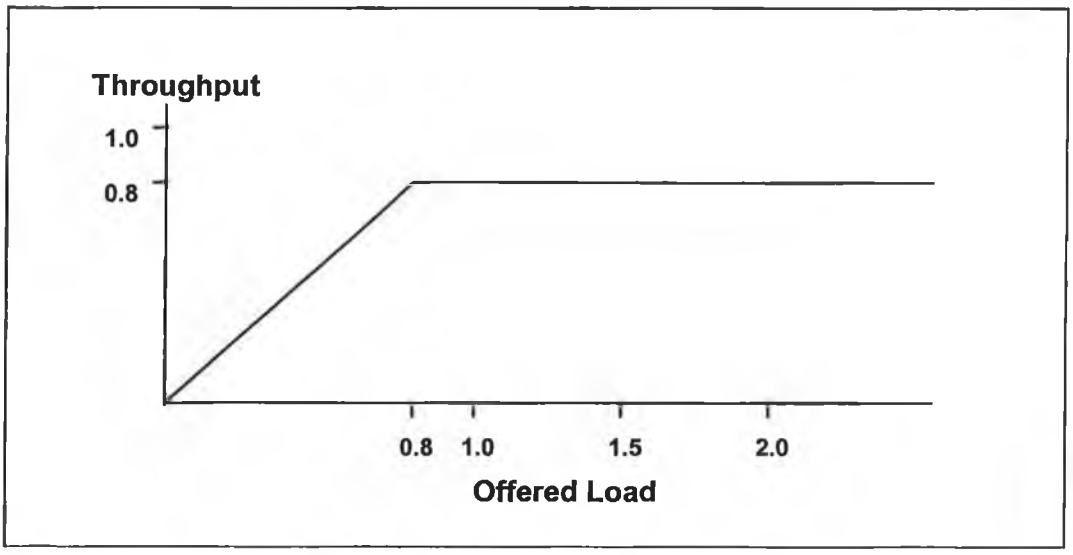


Fig. 2.7: Ideal Throughput Characteristics of a System under Overload

3. Processor load (i.e. the total work that the processor must do – including accepting calls, processing accepted calls and rejecting calls) should not exceed some threshold (usually set at either 80% or 90% of total processor capacity), even in the event of an extreme overload, to ensure that management and other non-switching functions (including congestion control routines) may be implemented [Sabourin91].

4. Average response times - the length of time a user must wait for a response from the system - must not increase noticeably due to overload. For example, in telecoms, this is both to comply with international performance standards on post-dialling delays (see [E.721] and [E.723]) and to minimise the number of call abandonments due to customer impatience (the impatience of both ISDN and IN customers is well documented - see [MacDonald94], [Bolotin94] and [Hoang90]). Call abandonment is highly undesirable, not only because resources which were spent processing calls which are subsequently abandoned are wasted, but also because abandoning customers have a tendency to reattempt their calls (see [Roberts79] and [Burkard83]), thus exacerbating the overload condition.

In meeting the above requirements, a strategy will need to compromise between throughput and response time, as they affect each other adversely. In order to keep throughput high during overload, as many calls as possible must be accepted, thus increasing the length of the input queue and the delay experienced by a request while those ahead of it in the queue are processed. On the other hand, to maintain response times at a minimum, the requests must spend as little time as possible in the input buffer, thereby reducing the number of calls in the buffer and the number

handled by the processor. These parameters are also dependent on the length of the input queue. If the input queue is short, response times will be shorter, but throughput will also be quite low. If the input queue is longer, throughput (and load) will increase, but so will the delay experienced by the user. The best possible compromise must be achieved between these parameters in order to maximise the efficiency with which each of the above requirements are met.

2.2.2 An Overview of Congestion Control in the PSTN/ISDN

Overload control has been widely investigated in the area of providing protection to telephone switches. In general, each switch in a network is responsible for protecting itself from becoming congested. This is done by detecting any of a number of occurrences that are recognised as constituting an overload situation and then implementing some measures to counteract the problem. These measures may involve:

- refusing all new calls outright (callers are cut off with minimum delay and minimum load is expending processing rejected calls),
- assigning priorities to different call types and selectively reject calls according to their priorities when overload occurs (this may involve either processor or trunk reservation – see [Lindberg88] and [Rajaratnam96] for examples of this),
- sending commands to surrounding nodes, instructing them to re-route call setup requests via other paths where switches are not congested (this is referred to as flow control and may be based on the use of pre-defined alternative routes or dynamic routing algorithms – see [Zepf91], [Dziong89] and [Langlois91] for examples).

Therefore, in overload investigations, it is usual to assume that the system to be protected consists of only one element - the switch.

The most popular model used for modelling systems under congestion (see [Korner91] and [Wallstrom91]) consists of a single processor with a controllable throttle at the input and a single feedback loop as a simplification of the delays (e.g. the time while the user is entering the desired digits or the conversation phase of the call) between processing times for each call (shown for a PSTN call in Figure 2.8 [Seraj85]). In all cases, irrelevant of which control strategy is in use, [Korner91] shows that the delay must have an exponential distribution (see Chapter 3, Section 3.2.3.2), because constant delays result in violent fluctuations in the processor load during operation. The algorithm for the detection of overload executes at the processor and when it deems that congestion has occurred and intervention is required, a message is sent to initiate the throttle algorithm, which then manages the input stream in order to reduce the number of calls to the system. Note that, since the distribution of the random arrival of calls at a switch tends towards the

Poisson distribution, an ideal Poisson generator is used to generate all new call requests. The graphical model is as shown in Figure 2.9.

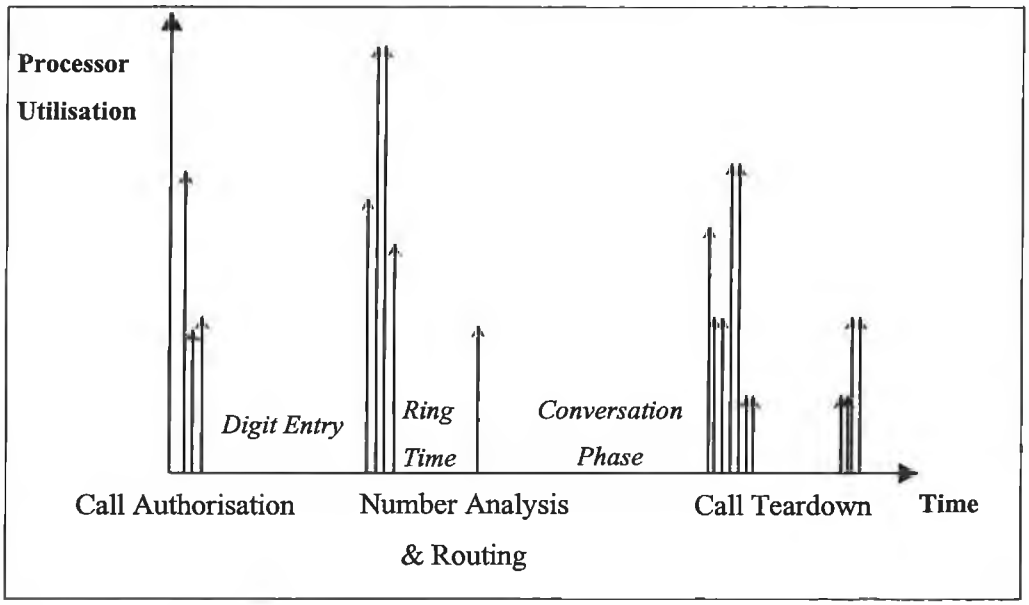


Fig. 2.8: Load Profile for a PSTN (non-IN) Call

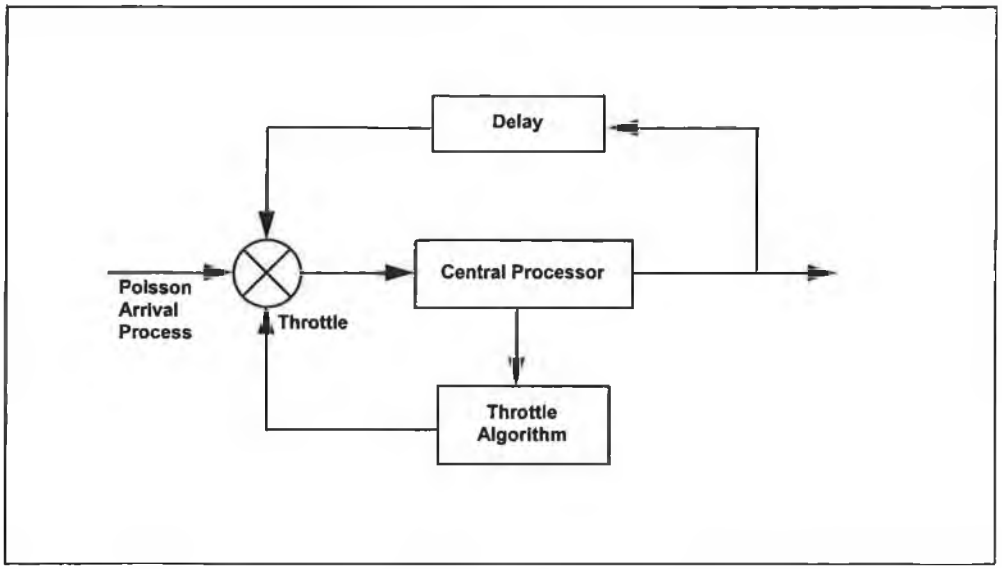


Fig. 2.9: Congestion Control Model of a Switching System

Note that this model is also suitable for monitoring distributed switching systems ([Manfield91], [Daisenberger85]), as, in general, these systems consist of a main central processor which sets up and controls all calls, and a number of peripherals which maintain line and trunk integrity. As it is the central processor that must be protected, the control algorithm resides here and the throttle is implemented separately at the peripherals. Therefore, the only difference in the model between single-processor and distributed systems is the location of the throttle algorithm, which, in terms of operation, has a negligible effect on system performance.

2.2.3 Classification of Switch Congestion Control Strategies

There are two types of congestion control strategy used in switch congestion control (both for Stored Program Controlled (SPC) and distributed systems). These may be classified as *active* and *reactive* strategies. An active strategy is always in place and permanently restricts access to the system, thereby preventing overload. Reactive strategies, on the other hand, only become active when the onset of overload is detected and ceases when the overload condition ends. Note that, for all strategies, when an incoming call is accepted, it is vital that it successfully completes processing to the point where termination (and charging) occurs - both for economic reasons and to optimise throughput. Therefore, any calls which have already received some processing *must* be given priority over any new calls which have not yet undergone any processing. This criterion ensures both that the call is either rejected immediately or serviced (i.e. not delayed and then rejected) and that there is no waste of processor time through blocking calls that have already received some attention.

2.2.3.1 Active Congestion Control Strategies

The principle active strategies used for managing congestion control in switches are Call Gapping, Window, Leaky Bucket and Token Throttling. These may be described as follows:

Call Gapping – This mechanism involves the use of a timer set to expire after a gap interval *g*. For each call arriving at the switch, the gap interval timer is checked. If the timer is inactive, the call is accepted and the timer is set. Until this timer expires, all further arriving calls will be unconditionally blocked. After the gap interval has elapsed and the timer becomes inactive, the first call to arrive is accepted and serviced and the gap interval timer is set again. This mechanism is illustrated in Figure 2.10 [Tsolas92]. Note that, while Call Gapping (CG) is described here as a switch congestion control strategy, it also has applications in network traffic management (e.g. [Turner91]).

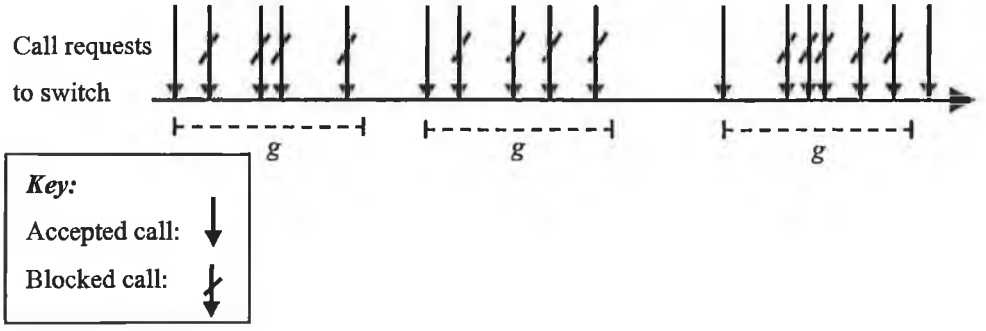


Fig. 2.10: Call Gapping Mechanism

Window Mechanism – Window is applicable only in distributed systems (the mechanism actually originated in client/server computing environments [Tsolas92]) and is used primarily in peripheral processors of distributed switches to protect the central processor. Here, each peripheral processor keeps track of the number of requests, w, which have been sent to the central processor and for which a response remains outstanding. Each time a new request arrives at a peripheral processor, its current value of w is compared to W (the *Window* value of the peripheral processor, with $1 \leq W \leq Wmax$, where $Wmax$ the maximum allowable Window size). If $w<W$, then the new request is dispatched to the central processor, w is increased and a timer is set for that request. This timer corresponds to the acceptable delay for the request at the central processor. If, on the other hand, $w=W$, the request is rejected immediately. Each time a response is received from the central processor, w is decreased. Each time a timer expires, its associated request is rejected, and both w and W are decreased. When a pre-defined number $Cmax$ of responses have been received from the central processor, W is increased. At the central processor, the length of time a request has been waiting for service is calculated before processing of the request begins. If the wait time exceeds the acceptable delay, then the request is abandoned. This mechanism [Tsolas92] is shown graphically in Figure 2.11.

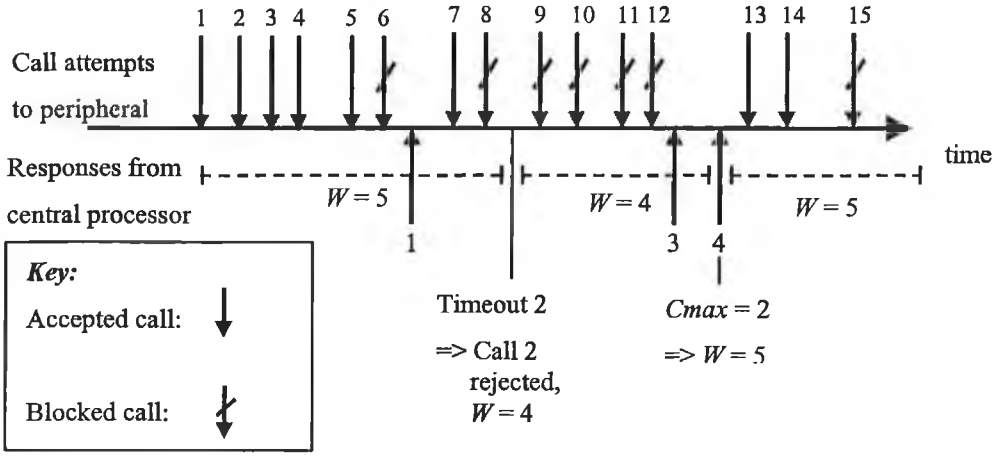


Fig. 2.11: The Window Mechanism

The algorithm described in [Manfield91] is an example of the Window mechanism applied in a distributed switch, with $W=Wmax=1$. Window is also widely used for congestion control of other distributed systems – namely, computer networks [Akyildiz90], telecoms networks (e.g. [Luan89] and [Doshi91]) and Intelligent Networks (discussed in Section 2.3).

Leaky Bucket – This is a traffic shaping mechanism with three parameters – a Leaky Bucket interval T_{LB}, a limit counter, S_{LB}, for the number of requests which may enter the Leaky Bucket during T_{LB} (i.e. S_{LB} is effectively the depth of the bucket) and a counter, K_{LB}, representing the

number of requests which may leave the bucket during T_{LB} (analogous to the size of the hole in the bucket). The graphical representation of this mechanism [Pham91] is shown in Figure 2.12.

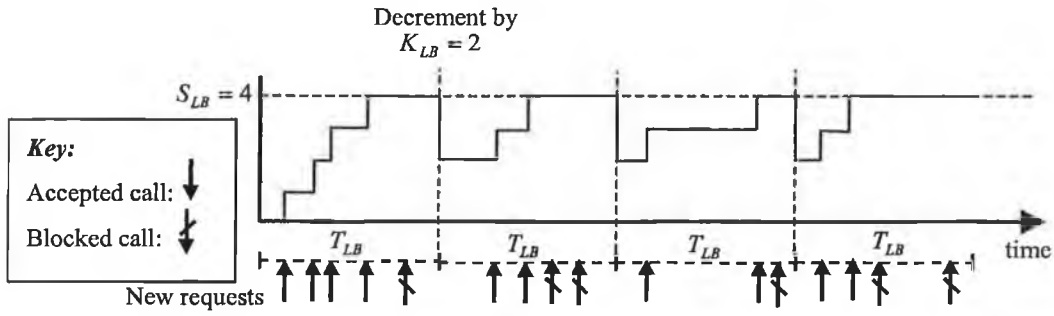


Fig. 2.12: The Leaky Bucket Mechanism

Token Mechanism – For the token algorithm, there are a fixed number of tokens in a token bank (queue of tokens). Each new request to the system must take a token to be accepted to the system. Those requests, which arrive when no token is present in the token bank, are rejected. When a request completes processing, its token is returned to the token bank. This is shown in simplified graphical form in Figure 2.13 and is described comprehensively in [Seraj85], as the primary congestion control strategy used in Ericsson AXB (data) switches.

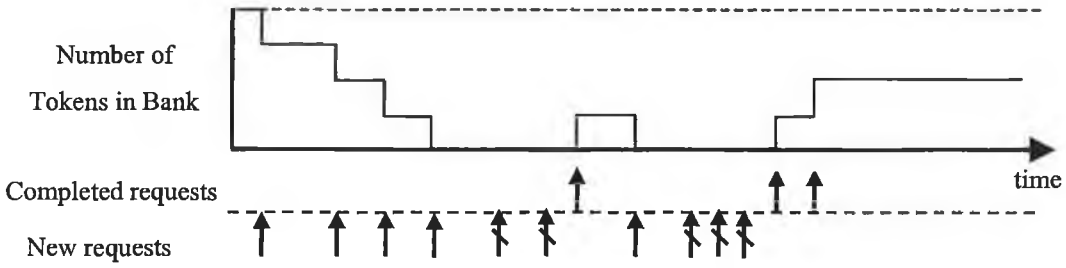


Fig. 2.13: The Token Mechanism

Note also that a hybrid of Token and Leaky Bucket (called Token Bucket) exists, whereby a parameter S_{TB} controls the depth of the token bucket and K_{TB} controls the rate at which tokens are released from the bucket.

A few comparisons may be made regarding the applicability of the active control strategies described above. Window is the only strategy described above which is directly applicable to distributed systems. Call Gapping, Leaky Bucket and Token are, as described, applicable to single systems. However, they may also be adapted to act as throttles in a distributed system, i.e. they may form part of a reactive system, where a remote detection method drives them by defining either the gap interval for the CG throttle, the number of tokens which should be made available in the Token throttle or the parameters of the Leaky Bucket throttle (described in Section 2.2.3.2).

Regarding resource requirements of the different strategies, Call Gapping is the most efficient strategy in terms of resource usage, as it requires only the maintenance of one timer. Leaky Bucket is only slightly more complex, requiring the maintenance of two counters and one timer. The Token mechanism, on the other hand, requires the maintenance of a queue of tokens (although this could be achieved more simply, using counters) and some mechanism for associating a token with a request. This means that Token requires more resources than Leaky Bucket. Window, however, has the largest footprint, as it requires, not only the maintenance of three counters, but also a timer for each request in the system.

Regarding operation of the strategies, active strategies generally tend, due to their fixed nature, to be unnecessarily restrictive as they occasionally cause calls to be rejected unnecessarily during small peaks in traffic load [Sabourin91]. Also, the fact that they are always in place makes them a continual drain on processor resources, which is highly undesirable. Therefore, it is more common to form a reactive strategy by using adapted versions of Call Gapping, Leaky Bucket and Token as throttles, in conjunction with a detection method, as described in the following section. Window, on the other hand, is not easily adapted to be a throttle and tends to be used as described above.

2.2.3.2 Reactive Congestion Control Strategies

As can be seen in Section 2.2.3.1, active congestion control mechanisms tend to have a single algorithm executing in a single location. A reactive overload control method, on the other hand, usually consists of two algorithms working together – namely, a detection routine and a throttling mechanism. The detection routine may be either continually active or may execute at set intervals. When executed, the detection algorithm recognises the presence of an overload situation, decides the level of overload and the suitable action to combat it and then sends a command to the throttle, which restricts the input to the system accordingly. Note that the separation between overload detection and response makes the reactive class of strategies very suitable for the congestion control of both single switches and distributed systems (distributed switches or networks of switches).

The range of detection routines that have undergone the most investigation includes:

1. **Queue Length Control (QLC)** – each time a request is added to a queue, the queue length is compared to a predefined maximum value, which, when exceeded, is classified as an overload condition.

2. **Load Measure Control (LMC)**- the mean load of the central processor over the course of a pre-defined time interval is estimated at the end of each consecutive interval (using algorithms

such as [Kallenberg89]) and compared with a maximum permissible value. When this value is exceeded, an overload is deemed to exist.

3. **Call Count Control (CCC) -** the number of new arrivals to the system is counted (old calls returning to the queue after a delay are excluded from this figure) over a pre-defined time interval and if it is found, at the end of the interval, to be greater than a specified maximum, alerts of an overload condition.

4. Any combination of the above - see [Wallstrom91] and [Villen85]. For example, CCC could be used in conjunction with QLC. CCC initially detects an overload condition, and its response is to lower the threshold of acceptable queue length. When the QLC algorithm detects that this threshold has been exceeded, the throttle is initiated.

Most reactive congestion control strategies are based on either the direct use of the above detection methods or on variations of these methods (e.g. [Daisenberger89]). The range of throttles defined to be used with the above strategies include:

1. **Call Gapping Throttle** - the detection routine sends an overload level to the throttle, defining a suitable *gap interval*, which is a length of time after a call has been accepted during which all new arriving calls are blocked. A *gap duration* may also optionally be included – this is the length of time for which the throttle should be in place. CG effectively places an upper bound on the acceptance rate of a system. This throttle is one of the primary throttles used in Intelligent Networks (described in Section 2.3), but also has applications in general network traffic management, as described in [Pham91].

2. **Percentage Throttle** – the detection routine sends an overload level to the throttle, defining the percentage of incoming calls to be accepted. All other calls are rejected. This is referred to as percent thinning (PT) and is commonly used in PSTN and ISDN switches.

3. **Token Throttle** – this is an adaptive version of the active token mechanism, in which the number of tokens available in the system is defined by the severity of the detected overload – see, for example, [Berger91a].

4. **Leaky Bucket Throttle** – this is an adaptive version of the active Leaky Bucket mechanism, in which the values of the limit counters are defined by the severity of the detected overload – see, for example, [Pham91].

Each of the three detection strategies described here- QLC, LMC and CCC - has its own advantages and disadvantages when applied to a single-queue model. These are detailed in [Korner91] and may be explained as follows.

Queue Length Control: QLC responds almost immediately to the onset of congestion, as the input queue length is monitored continuously. However, recovery time is slow and there are heavy fluctuations in the load. The reactive nature of QLC means that it is not possible to achieve a satisfactory trade-off between response times and throughput - low response times can only be achieved through maintaining the queue length threshold low, thus resulting in reduced throughput.

Load Measure Control: Fluctuations in queue length and load for LMC are smaller than for QLC. However, due to the fact that call acceptance only takes up 30% of total call processing time, calls accepted during an interval may require more load during a later interval. Therefore, the processor sees traffic levels as being greater than they actually are, resulting in overload being detected well before its actual occurrence (i.e. when input traffic is only approximately 60% switch capacity) – see also [Sabourin91]. LMC is also slow to recover from congestion. This is because the measured load is defined by calls that have already been accepted. Therefore, by the time overload is detected, many calls have already been accepted, and recovery is quite slow. Note also that the efficiency of this method is very dependent on the length of the control interval - if the interval is too short, overload controls will be initiated even earlier, whereas if it is too long, very many calls will have been accepted and recovery will take even longer.

Call Count Control: CCC monitors the number of newly arriving calls and therefore gives an accurate reflection of the input traffic. This means that it responds very quickly to the onset of overload - in fact, due to the random nature of incoming traffic, CCC tends to respond prematurely to overload - when incoming traffic is at 75% processor load. This means that overload is not very serious when throttles are put in place and therefore recovery is also very fast, with minimal fluctuations in input queue length and load. Note that, as with LMC, quality of operation is dependent on control interval length - in a short control interval, there will be a large variation in arrival intensity and controls could be initiated unnecessarily, while many calls will arrive during a long interval and therefore, recovery will be slower.

The conclusion to be drawn from the above is that, for a single-queue system, CCC is the most effective overload detection method. It responds quickly to the presence of congestion and recovers quickly while maintaining good levels of throughput. However, it remains to be seen if CCC remains the best technique when applied to the Intelligent Network. The IN is not structured as a network of single-processor systems and also presents a number of new issues in the area of

congestion control (detailed in Chapter 1). This means that it is impossible to draw any conclusions, at this time, as to which detection method would be most effective at protecting the SCP from overload.

Regarding throttling mechanisms, percent thinning is the easiest to implement and requires the least processing power as it requires only the maintenance of two counters. Leaky Bucket has a simple algorithm and requires two counters and one timer. CG is also simple to implement and requires the maintenance of one (or possibly two) timers. The Token mechanism is very complex to implement (see [Seraj85]) and requires the maintenance of a few queues of tokens. All throttles mentioned may become more complex if it is required to allow variation in the throttling levels applied to different call types or calls coming from different source nodes. No conclusive comparisons are available between these throttle types in SPC/distributed switch/traffic management studies, and therefore, their applicability in IN congestion control should be directly established using IN models. Comparisons are available in the IN domain and will be described in Section 2.3.

2.3 IN Congestion Control

2.3.1 A Description of the Models used in IN research

As the IN is made up of a number of different PEs that interact to execute a service, the issue of congestion prevention and control is a much more complex one than for simple switching systems. In the IN, as the SCP is most central to correct operation, protection of it is of paramount importance. Therefore, to date, all documented research in the area of IN congestion control has been focused on developing ways to maximise SCP efficiency.

Two different types of models have generally been used. The first is very similar to the single-processor model used in the investigation of switch operation. However, in this case, the model represents the SCP (see Figure 2.14). Input streams are usually assumed to be Poisson in nature and the times between SCP processing of a single request (i.e. when SDP or IP access is required) are represented as a feedback delay of arbitrary length. The throttle is placed at the input to the system, so that only new requests may be rejected, but it is not specified whether it is integrated into the SCP or located elsewhere in the network. Examples of research involving the use of this type of model include [Smith95] and [Nyberg95b]. Its use is understandable in that, although the model is an oversimplification of the real system, it is very useful for mathematical analysis of SCP operation.

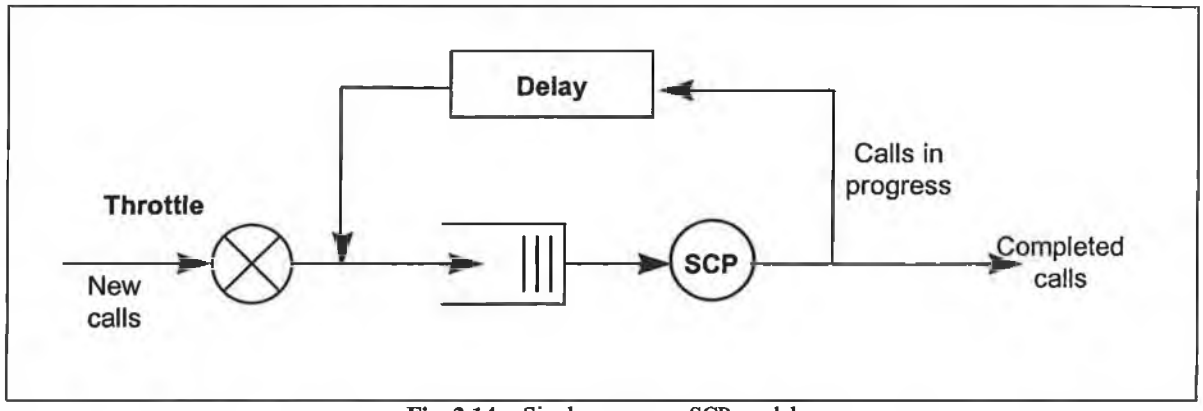


Fig. 2.14: Single processor SCP model

The second type of model is more widely used as it more accurately represents the elements of an IN, in that it contains multiple SSPs and one (or more) SCP. However, due to its added complexity, mathematical analysis of the system is no longer a simple process. Therefore this model is primarily used in simulations.

In general, the model appears as in Figure 2.15, with multiple SSPs connected to one SCP (e.g. [Nyberg94], [Pham92], [Kihl95], [Rumsewicz96], [Yan94], [Kwiatkowski94a]). In some cases, a Service Transfer Point is also included [Galletti92] to represent interactions with the SS7, which would be the transmission medium between elements in a real network, although when the SS7 is partially represented like this, the transmission delay is still only represented by a constant value. Occasionally also, a combined Service Switching and Control Point (SSCP) is included, and is represented as a delay queue in series with a job queue – see [Kihl97]. [Leever93] and [Kwiatkowski94b] provide simple analytical method to calculate load and service delays for this model. [Newcombe94] extends this model and solves mean delays using the decomposition method (described in Chapter 3).

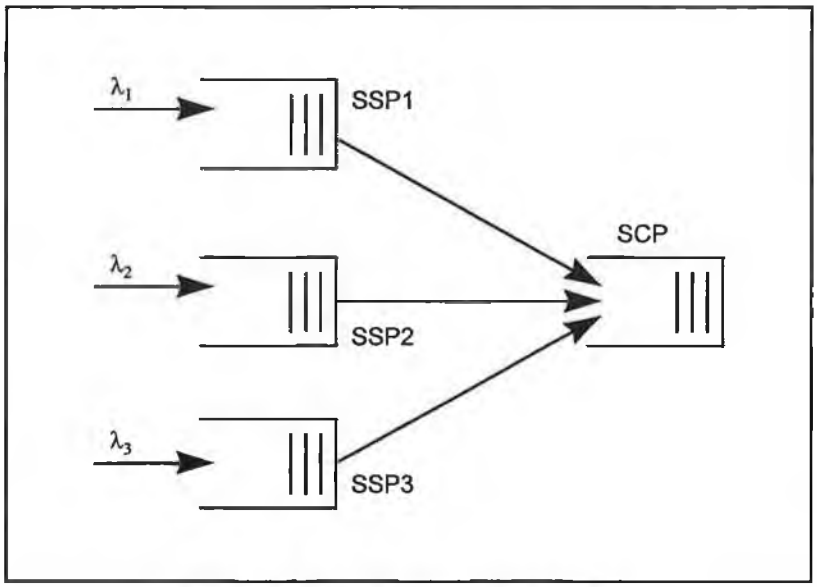


Fig. 2.15: Multiple queue model of the IN

Generally, different research studies using this model make a number of assumptions. These include:

- It tends to be assumed that all calls in the network are of the same type, or if of different types, have the same load profile. Information flows in the system tend to be simplified. For example, in [Galletti92], all traffic is unidirectional - i.e. the SSPs send requests to the SCP and the SCP does not respond. In most other investigations, communication between the SSP and SCP is made up of a single query/response pair (e.g. [Pham92], [Kihl95], [Tsolas92], [Nyberg94] and [Hebuterne90]) in which the SCP receives a query from an SSP, processes it and returns a response to the relevant SSP. This is generally either represented either as a simple number translation service (e.g. Freephone, which does only require one SCP transaction) or as the first transaction in a more complex service. Justification for this simplification tends to be that all service requests should be rejected or accepted during the initial transaction and accepted calls should complete successfully, and therefore, as overload only impacts on initial transactions, only these need to be modelled. Some research studies (e.g. [Rumsewicz96], [Pham92]) do include information flows for more complex services (e.g. services which require processing at the SCP on multiple occasions), but again, only the impact of the initial service transactions are investigated. This is, however, not completely accurate, as it does not take into account the effects of the load profiles of the different request types in the system (i.e. how the necessary processing of old calls at the SCP impacts on congestion situations).
- It also tends to be assumed that all service types in the network have the same priority level, i.e. that during overload, all requests may be throttled equally. An exception is [Lee97], who did examine how CG could be altered to cater for different call priorities.
- Retrials and their effects on congestion control strategies are generally not investigated in most studies. An exception is [Manfield91], who included retrials in his study of overload control of hierarchical switching systems.
- Many studies provide different interpretations as to the meaning of "fairness". For example, [Lee97] classifies fairness as "the probability of rejection ought to be the same for all the subscribers, irrespective of which SSP they are connected to", while [Hac98] claims that fairness means that only sources (SSPs) which are causing overload should be targeted. Note that theses definitions of fairness are contradictory – however, the first definition (i.e. that all subscribers to a particular service should be treated equally) is more generally accepted and may be referred to as "subscriber fairness". [Rumsewicz96], [Galletti92] and [Tsolas92] say that a fair strategy should target only the service type which is causing the overload (a form of Focussed Destination Overload Control (FDOC)) – this may be termed "service fairness".

- In general, traffic arrival at SSPs is assumed to be Poisson in manner and SSPs are modelled as infinite First-In-First-Out (FIFO) queues with either no delays, constant delays or exponentially distributed services times.

Intelligent Network congestion control strategies tend to be based on the common switch-based strategies described in Section 2.2. Active congestion control strategies are located in the SSPs and are communication-less – i.e. they do not require notification of overload from the SCP. Reactive strategies, on the other hand, are communication-oriented - the detection algorithms are located in the SCP and send overload notification to the throttles in the SSPs. In both cases (active and reactive), all rejections of service requests take place at SSPs. Justification for this is simple - SCP processing time is at a premium during an overload situation and should be maximised in the execution of services. Intuitively, therefore, implementing a throttle to reject requests at the SCP is a waste of valuable processor time and should be avoided. Also, a substantial part of call processing time at the SCP is spent unmarshalling the call (i.e. unwrapping the SS7 protocols) before it may be interpreted and a decision made as to whether it should be accepted or rejected [Korner94] – this overhead, in terms of wasted capacity, is unacceptable and therefore calls should be rejected remotely. Therefore, throttles are always situated in the SSPs and restrict traffic to the SCP according to the control data passed to them by the SCP. In general, this control data tends to be an overload level, a call gap interval or (in cases where PT is used as the throttle) a percent thinning coefficient.

Note that a form of the CG throttling mechanism described in Section 2.2.3.2 has been standardised by Bellcore as part of the Advanced IN (AIN) standards [Bellcore92]. This mechanism is called Automatic Code Gapping (ACG) – automatic, because it is possible to dynamically control the gap interval level to be applied by the throttle. This is not, however, as adaptive as it sounds, as the standard includes a fixed table of permitted gap intervals from which an appropriate interval must be chosen. ACG throttles may be put in place selectively at different SSPs by either the SMS (SMS-Originated Code Control (SOCC)) or the SCP (SCP Overload Control (SOC)). SOC includes a parameter that enables the selective restriction of calls based on the first six digits of the originating/terminating number, while SOCC also allows the specification of parameters that permit selective control of incoming calls (e.g. by service type). Note also, however, that the ACG specification is specific to the AIN standards, and no such specification exists in ITU CS-1. Therefore, there is no requirement to include this specification in our CS-1 IN model and, as will be seen in Chapter 4, the version of CG implemented in the model conforms with the (table-driven) throttle description provided in Section 2.2.3.2. In Chapter 6, a new, fully adaptive CG throttle is defined – the motivation for this is provided by [Smith95], who demonstrates both that the gap intervals provided in the AIN standard are ineffective, and that an adaptive CG throttle has behaviour far superior to that of a fixed table-driven CG throttle.

Some variations on the more usual active and reactive strategies have also been investigated in the IN arena. For example, [Hac98] describes a hybrid Window/Adaptive CG strategy for IN protection, [Nyberg95a] examined the use of Proportional Integral Differential (PID) controllers for IN overload control and [Galletti92] and [Rumsewicz96] both propose extensions to usual strategies to facilitate FDOC. Note also that the use of optimisation techniques for performance management is not unusual. Some examples include [Pham91], who uses revenue optimisation to dynamically define Leaky Bucket parameters, while [Milito91] uses revenue optimisation to decide if a newly arrived call to an IN SSP should be blocked. [Angelin95] and [Arvidsson96] investigate the use of profit optimisation to decide, based on predictions of SS7 and processing delays, whether a new call request should be accepted at an SSP. The efficiency and performance of these less common strategies are not compared to that of the common strategies. However, a number of papers do exist which cover the comparison between:

- CG, Leaky Bucket and PT throttling mechanisms (no study has been published which examines the performance of Token active strategies or throttles in the IN context),
- most common active and reactive IN congestion control strategies.

2.3.2 Comparison between Throttles for the IN

Leaky Bucket vs CG: [Pham91] suggests that the Adaptive Leaky Bucket throttle performed better than the Adaptive CG throttle, primarily due to the fact that it is less strict and can handle bursty arrivals because it limits the number of calls which can be accepted within a period while CG only accepts strictly one call per period. However, this claim is not well supported in the paper, and no other references were found to support this position. On the other hand, [Lee97] compares a Continuous Gapping throttle (analogous to an adaptive Leaky Bucket strategy with a leak rate of one request per interval) to the normal adaptive CG throttle and established that both throttles perform equivalently, if suitable gap intervals are defined in each.

Percent Thinning vs CG: Considering the operation of the two throttles, [Kihl97] found that they were generally equally efficient at protecting the SCP (providing the CG intervals were appropriate to the network topology and call arrival rates). However, PT exhibits subscriber fairness, as the SCP notifies all SSPs to reject a certain proportion of their arrival traffic and therefore all SSPs are throttled equally [Rumsewicz96]. By extension of this, PT is scalable, as the PT coefficient is not dependent on the size or number of SSPs in the network or on the arrival rates to each SSP [Berger91b]. CG, on the other hand, puts the same gaps in place on all SSPs, irrespective of size, with the result that larger SSPs (or SSPs with greater arrival rates) are more heavily throttled (i.e. it is not subscriber fair). CG is also not scalable. However, it is more robust to changes in arrival rates (as it puts a firm upper limit on the number of calls which may be

accepted [Berger91b], [Hebuterne90]) and also tends to throttle services with greater arrival rates more heavily, resulting in a certain implicit level of service fairness.

2.3.3 Comparison between Active and Reactive Strategies for the IN

The most popular active congestion control strategy used in IN implementations is Window. Reactive strategies tend to use QLC, CCC or LMC as the detection method in the SCP, and either CG or PT in the SSPs. [Pham92], [Tsolas92], [Nyberg92] and [Nyberg94] carried out comparative studies between active and reactive IN congestion control strategies. Unfortunately, it was found that the results differed paper to paper. [Pham92] concluded that Window was generally superior to CG, with one of the primary reasons for this being that Window is updated as to the state of the SCP every time a response is received (i.e. it has a tighter SSP-SCP control loop than reactive strategies). However, this conclusion may be questioned, as the monitoring interval used by the CCC/CG strategy was 20 seconds – which is far greater than the optimal, as specified by [Korner91]. Pham also found that Window was fairer than the reactive strategy in terms of rejection rates at different SSPs in the network – however, it is unclear whether this is actually a function of reactive strategies in general, or of the fact that CG was used as the throttle, where CG has already been recognised as being quite unfair (see Section 2.3.2 above). [Tsolas92], on the other hand, found LMC/CG to be superior to Window. He found that (with a shorter monitoring interval than that used by [Pham92]) the reactive strategy responded more quickly to the onset of congestion. LMC/CG was also considered to be more flexible, as it facilitated the selective throttling of services or sources. Both [Nyberg92] and [Nyberg94] found reactive strategies superior to the Window strategy for similar reasons to [Tsolas92], although it is interesting to note that [Nyberg94], who used PT instead of CG as the throttle, added that the reactive strategy was fairer than Window. However, the fact that no definitive answers are available in existing literature means that it is still debatable which method is superior.

2.4 Conclusions

To conclude this chapter, we summarise that the most popular IN congestion control strategies in place today are:

- Window, an active strategy located in SSPs,
- Various reactive strategies, consisting of either CCC, LMC or QLC as detection methods at the SCP, working in conjunction with either a CG or a PT throttle in the SSPs.

To date, a number of studies have been conducted into comparing the operation of Window with that of various reactive strategies, but no definitive conclusion has been drawn as to which is best. Also, while CCC has been established as the best detection method in SPC systems, it remains to

be seen whether it is still the best in the IN arena. Research into throttle types have found that both CG and PT have advantages and disadvantages, so it is unclear which is superior in a practical IN implementation. It was therefore decided to make no assumptions regarding superiority of strategies, algorithms or methods at this point.

Chapter 4 will begin by focusing specifically on establishing the effectiveness of various detection methods used in conjunction with the CG throttle. It will also be necessary to find out how adaptable these detection methods are in order to meet the added requirements of the IN. Each of the detection methods described above will therefore be implemented on a new type of model of an IN system and executed in conjunction with the CG throttle in order to see which one best meets the needs of the SCP. The detection algorithms will be located at the SCP, and will interact with a CG throttle in the SSP in order to control the quantity of traffic arriving at the SCP. When the best detection method has been established, this will be used in conjunction with both CG and PT to establish the best common reactive strategy. When the best solution for a reactive strategy has been established, its operation will be compared with that of the active Window mechanism. The results of Chapter 4 will be twofold:

- the best of the most commonly used strategies will be identified (based on the principles described in Section 2.2.1 of this chapter) for IN SCP overload control,
- the limitations of these strategies will also be identified, which will help clarify the requirements on an ideal IN congestion control strategy.

Then, in Chapter 5, the IN model will be enhanced to include non-IN calls, multiple finite-capacity SSPs (i.e. SSPs which may experience overload), multiple traffic types with different load profiles and priorities and a new strategy will be developed which will allow the efficient performance management of this new, much more complex IN system. Throughout this work, the criteria that will be used to evaluate the validity and efficiency of the developed congestion control strategies are:

- **SCP queue length**: It is vital to ensure that the SCP is protected from overload at all times. A good way of estimating the dynamic load presented to the SCP is to monitor variations in the input queue length - the occurrence of any overload situation will immediately be reflected here.
- **SCP throughput**: The ideal congestion control strategy will protect the SCP at all times, while maximising its throughput. The SCP queue length statistic will provide information as to the quality of the evaluated strategies at protecting the SCP, but in order to ensure that the strategies are not excessively harsh, their effect on SCP throughput must also be estimated.
- **SSP load and throughput**: For the initial study of SCP overload control strategies (in Chapter 4), SSP capacity is modelled as being infinite and therefore, SSP throughput is not

relevant. However, the enhanced model presented in Chapter 5 will include finite-capacity SSPs that are also prone to overload and the congestion control strategy to be developed will need to take into account the efficiency of the SSP.

- **User delays**: The average length of time each user must wait for service processing to be completed. In general, network processing time of user requests for service should not exceed 2 seconds (as after this time, users will begin to abandon calls [Kant95]), although, in cases where IP processing is required, longer delays are acceptable, as they include times when the network is interacting with the user (i.e. when the user is busy and therefore not impatient). As delays will vary for each service type, depending on the processing required, it is necessary for each service to be monitored individually.

Chapter 3

Analysis Tools & Methods

In this chapter, the background information required to understand the ideas presented in Chapters 4, 5 and 6 is presented. Firstly, OPNET, the tool used in the development and simulation of the IN model, is described. Then, in Section 3.2, the queuing theory used in the specification of the IN analytic model are explained. Section 3.3 finally presents the theory behind mathematical optimisation and linear programming.

3.1 An IN Simulation Tool

3.1.1 Using OPNET for IN Simulation

In order to ascertain the behaviour of the Intelligent Network under various congestion control strategies, it is necessary to develop both a simulation model of the IN (made up of service traffic sources and a high-level model of the IN which deals with the traffic in a manner similar to a real IN) and an analytical model (a mathematical model which allows the mean state of each element of the IN to be evaluated for various mean traffic arrival rates). For the simulation, it was decided to use the OPtimised Network Engineering Tools (OPNET) package to develop the IN model. OPNET is a hierarchical object oriented simulation tool, designed specifically for the development and analysis of communication networks. It provides a graphical interface to the user for the specification of models. The models of protocols and algorithms employ a hybrid approach by allowing the user to embed C language code within a graphically laid out finite state machine. The specification of processes in C is facilitated by an extensive library of support functions, which provide a wide range of simulation services. It also provides a set of analysis tools to interpret the simulation results in graphical form.

There were a number of reasons for choosing to use OPNET for modelling the IN system. These included the following:

- OPNET's hierarchical nature simplifies the design of complex systems, through the separation of concerns into network, node and process levels.
- OPNET allows the creation of multiple instances of nodes, thereby making it easy to scale the IN (in terms of the number of SSPs). Each of the node types of the IN (the SCP, SDP, IP and SSP) have to be developed only once, but the overall system may consist of multiple instances of each.
- Library functions are provided for the construction of queues and various traffic arrival time and service time distributions.
- The core parts of a system (i.e. the processes) are event-driven finite state machines, thus allowing the modelling of real-time systems, such as the IN.

- The programming language underlying the application is called Objective C - this is effectively standard ANSI C, with a large number of OPNET-specific library functions. Familiarity with C therefore reduces considerably the learning curve involved with using OPNET.
- Unlike a number of other modelling packages available, OPNET provides comprehensive and flexible support for acquiring statistics from a simulation and representing them in graphical form. It also provides various mathematical filters to facilitate rigorous analysis of statistics, which was a useful feature in the course of this work.

3.1.2 Operation of the OPNET modelling tool

OPNET simulations are based on four separate modelling domains called the Network, Node, Process and Link domains. The dependencies between these modelling domains are shown in Figure 3.1 below. As illustrated, network models rely on the definition of the node models, which in turn incorporate process models. In addition, link models are used to characterise links between nodes in the network domain.

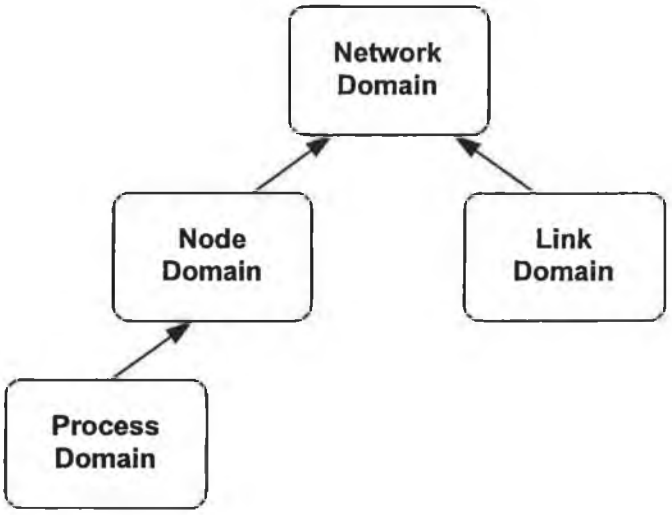


Fig. 3.1: OPNET modelling domains

In the Network Domain, node models are instantiated and each instance may be assigned independent attributes including identification, position, and user-defined attributes. Nodes that are designed to attach to physical links (i.e. which contain receivers or transmitters) may be interconnected to form arbitrary network topologies.

The Link Domain allows incorporation of custom or user-specified link models within OPNET simulations. These models are specified in C and are linked into the simulation. Point-to-point links are represented by lines between source and destination nodes. The point-to-point links are unidirectional, so a duplex link must be represented by two links, one for each direction. The

point-to-point links have a number of built-in attributes that can be specified by the user. They include the transit delay incurred by packets forwarded over the link, and the bit error rate - the probability of bit errors in packets transmitted over the link.

In the Node Domain, the internal structure of the nodes is defined. This structure consists of modules that can generate, process, store, receive and transmit packets and manage resources according to a user-defined process. These modules can be interconnected to form arbitrarily complex node architectures. A number of standard module types are available within OPNET and may be used directly or amended at process level to function in a user-defined way. The standard module types include:

- The *Ideal Generator* module, which provides a convenient stochastic packet source. The frequency of packet arrivals and the length of packets can be controlled by selecting any one of a range of probability distributions. The packets generated can also have a packet format specified, in terms of the fields within the packets and the information they hold.
- The *Queue* module, which incorporates C code and simulation kernel procedures to model processing functions of the node. The queue module may contain a number of subqueues, each of which can hold a list of packets. The queuing discipline used, the number of subqueues needed and the capacity of each subqueue can also be specified. It is also possible for the user to define whether the queue is active (i.e. has an in-built server, which removes packets from the queue and processes them) or inactive. If an active queue is specified, the service rate of the internal server may be defined by the user.
- The *Processor* module, which carries out set operations on any received packets. A range of processor types are available, although the most commonly used one is the sink, which is responsible for destroying packets and deallocating the memory assigned to them.
- The *transmitter* and *receiver* modules are used for communicating between nodes. A transmitter module of one node is connected to a corresponding receiver module at the destination node via a point-to-point link between nodes in the network domain. The maximum data rate for each of these modules can be specified.

Process models are specified using a graphical editor that captures the structure of the process in the form of a finite state machine (FSM). The FSM models a communications process by responding to changes in its inputs, modifying its state and producing new outputs. Process models may make use of a library of kernel procedures that support access to packets, network variables, statistic collection, packet communication and other simulation services.

The two fundamental components of an FSM are *states* and *transitions*. States can be used to represent the significant modes of the process and may have certain actions associated with them. An FSM can implement actions both on entering and on leaving a state. All states can be

considered as consisting of three different phases of traversal as shown in Figure 3.2 below. The first phase is the enter executives, which are always implemented on arrival to the state. The second is a possible resting phase and the third phase is the implementation of the exit executives.

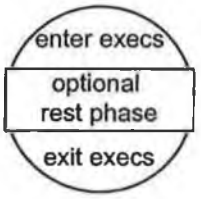


Fig. 3.2: Representation of a State

Two types of states are distinguished in OPNET process models - *forced* and *unforced* states. Forced states bypass the rest phase and proceed immediately to the exit executives. Unforced states, on the other hand, always cause the operation of the FSM to be suspended immediately after the enter executives have completed. An FSM will remain in the rest phase until a new interrupt is delivered to the process model, causing a transition to the exit executives. In fact, interrupts are absorbed by process models ***only*** when their FSMs are in a blocked condition, and thus necessarily occupying an unforced state. Therefore, unforced states are only used when it is required for the process to wait for a particular event to occur, the result of which is the generation of the appropriate interrupt for which the process is waiting. When the correct interrupt is received, the FSM will then leave the rest phase and start processing the exit executives. Execution will then continue until the rest phase of another unforced state is reached. The graphical description of forced and unforced states is shown in Figure 3.3.

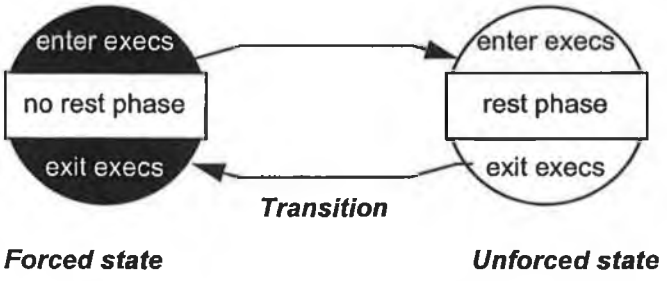


Fig. 3.3: A sample FSM

The transitions shown in the diagram represent the possible migrations between states. A transition is made up of a path description with an associated expression. When the exit executives have completed for a state, the transition expressions are evaluated (as boolean), to determine which transition should be followed and which new state entered. Since the finite state machine may occupy only one state at a time, only one transition statement should evaluate as true at any one time.

Within the process domain, it is possible for a user to create a new FSM with particular operations defined in C-code in the enter and exit executives of the states. However, it is more common, and much simpler, for a user to begin with the process model of a predefined module and amend its functionality as required. The generator, queue and processor modules allow existing code to be altered or enhanced in all executives, resulting in the creation of a new process type. The receivers and transmitters, however, do not permit user-access to the process level code. Any manipulation of packets at transmission time must therefore be carried out in other processes that may be connected directly to the relevant module at node level.

3.2 Analytical Network Modelling

3.2.1 Probability Theory

Probability theory concerns itself with describing random events through the identification of patterns in collections of related random events. As a simple example, if one were to toss a coin once, the outcome would be unknown (aside from knowing that it will be either heads or tails!). However, if the same coin was tossed one thousand times, it would be reasonable to expect that approximately 500 heads and 500 tails would result. This is the basic premise of probability theory - that accurate statements may be made about large collections of random events.

In order to analyse a given problem domain using probability theory, some terms must first be defined:

- A **sample space** Ω is the set of mutually exclusive exhaustive outcomes (**sample points**) of an experiment on a given random problem domain.
- In this context, an **event** is the result of a single random experiment and comprises some set $\{\omega\}$ of the sample space.
- A **probability measure** P of an event A is a measure of the likelihood of the occurrence of that event. It is measured in real numbers, where $0 \leq P[A] \leq 1$.

To give an example of these ideas, examine the behaviour of a die. Here, the sample space for the tossing of a die is {1,2,3,4,5,6}, i.e. the set of all possible outcomes of the toss. Let an event A be defined, for which the result of a toss of the die is 2. The probability of event A occurring is obviously $\frac{1}{6}$, as the die is equally likely to land on each face. Let us define another event B, for which the result of the toss of a die is less than 3. Two sample points satisfy this event, namely 1 and 2. Therefore the probability of B occurring is:

$$P[B] = \frac{\text{the number of sample points which satisfy the event}}{\Omega} = \frac{2}{6} = \frac{1}{3}$$

Note that $P[\Omega] = 1$.

3.2.2 Random Variables

Given this information, it is now possible to define the important concept of a random variable. A random variable (RV) is a real-valued function defined on a sample space Ω, i.e. it is a variable whose value is defined by the outcome of a random experiment. Mathematically, a real number $X(\omega)$ may be represented as the value which the random variable X takes on, when the outcome of the experiment is ω. A random variable may also be classified as a discrete or continuous RV, according to whether its **range** (the set of values which can take on) is discrete or continuous.

Now the **probability mass function (pmf)** of a **discrete** random variable may be defined as:

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

where X is a RV and x is a real number. Note that $\sum_{x_i} p(x_i) = 1$.

Also, the **cumulative distribution function (cdf)** of a RV X (both **continuous** and **discrete**) is defined by $F(x) = P[X \leq x]$. Note that:

$$\lim_{x \to +\infty} F(x) = 1 \text{ and } \lim_{x \to -\infty} F(x) = 0$$
$$P[x\ \pi\ X \leq y] = F[y] - F[x]$$

As an example of these concepts, a RV $X(\omega)$, which is dependent on the outcome of the toss of a die and has a range {-1,0,1}, is introduced. Let the value of –1 be the probability that the toss of the die results in a 1 or a 2, i.e. $P[X = -1] = P[\omega \in \{1,2\}] = \frac{1}{3}$. Further, let $P[X = 0] = P[\omega = 3] = \frac{1}{6}$ and $P[X = 1] = P[\omega \in \{4,5,6\}] = \frac{1}{2}$. The pmf of this random variable is shown in Figure 3.4.

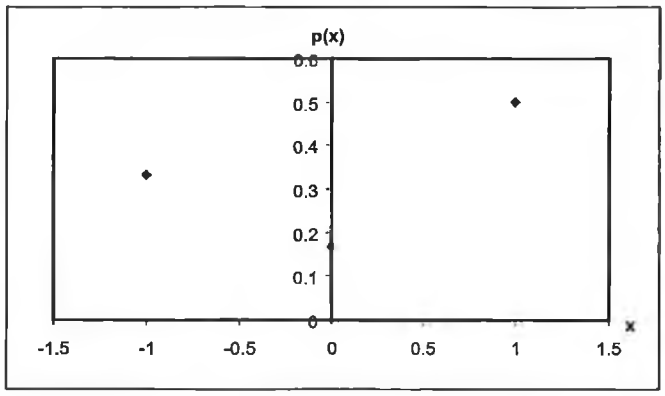


Fig. 3.4: pmf of $X(\omega)$

The cdf of $X(\omega)$ is as shown in Figure 3.5.

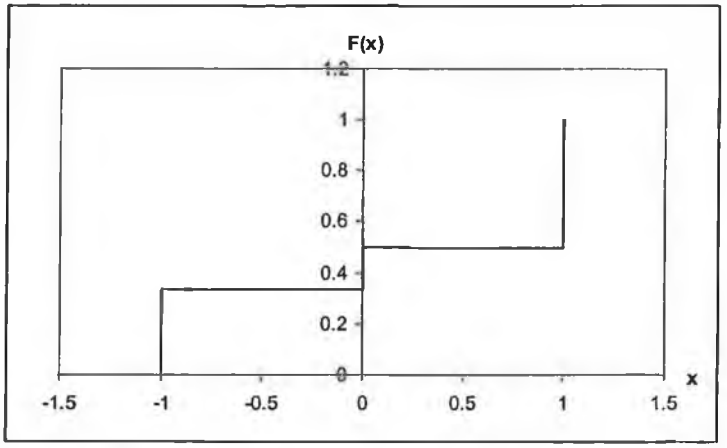


Fig. 3.5: cdf of $X(\omega)$

Finally, the probability density function (pdf) of a **continuous** RV X (defined by $p(x) = 0$) may be found, at each point x where f is continuous by:

$$f(x) = \frac{dF}{dx}$$

Note that:

$$f(x) \geq 0 \text{ for all real } x$$
$$\int_{-\infty}^{+\infty} f(x)dx = 1$$
$$f \text{ is integrableand } P[a \pi X \leq b] = \int_a^b f(x)dx$$
$$F(x) = \int_{-\infty}^{x} f(t)dt \text{ for each real } x$$

3.2.2.1 Moments of a Random Variable

The $\mathbf{k}^{\text{th}}$ moment of a discrete RV is defined by:

$$E[X^k] = \sum_i x_i^k p(x_i)$$

while the $\mathbf{k}^{\text{th}}$ moment of a continuous RV is defined by:

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

Two moments of a RV have been given special names. When **k**=1, this is the **mean** or **expected value** of a RV. When **k**=2, this is the **variance** of a RV.

The formula to calculate the expected value of a random variable X defined by a discrete sample space is:

$$\text{E[h(X)]} = \sum_{i=1}^{n} h(x_i)p(x_i)$$

When X is continuous, $E[h(X)] = \int_{-\infty}^{+\infty} h(x)f(x)dx$. Note that, when two random variables X and Y are independent, expectation is a linear operator i.e. that $E[XY] = E[X]E[Y]$ and $E[X + Y] = E[X] + E[Y]$.

The variance of an RV X, defined on a discrete sample space, is given by the formula:

$$V[X] = \sigma^2 = \sum_i (x_i - E[X])^2 p(x_i)$$

If X is continuous, its variance is given by $V[X] = \sigma^2 = \int_{-\infty}^{+\infty} (x - E[X])^2 f(x)dx$. Note that $V[X] = E[X^2] - E^2[X]$ and that for independent RVs, variance is a linear operator, i.e. $V[\sum X_i] = \sum V[X_i]$.

Knowing the expected value and variance of a random variable, the **square of the variation coefficients** (svc) of the RV may be calculated, according to $Kx = V[X]/E^2[X]$.

3.2.3 Random Processes

A stochastic (random) process is a function of two arguments - time and a probability space, and is therefore denoted by the term $X(t,\omega)$. For a fixed value of t, $X(\cdot,\omega)$ is merely a function of the probability space Ω - i.e. is a random variable. For a fixed value of ω, $X(t,\cdot)$ is a function of time and is referred to as a ***sample function*** of the process. Examples of random processes include:

- The number of call requests that can arrive at a switch in [0,t) is a discrete-state, continuous-parameter random process.
- The waiting time of an inquiry for processing is a continuous-state, continuous-parameter process.
- If $\{X_n, n = 1,2,..,7\}$ denotes runtime of a job, where n is the day of the week on which the job is running, this is a continuous-state, discrete-parameter process.
- If $\{X_n, n = 1,2,...,365\}$ describes the number of jobs per day of the year, this is a discrete-state discrete-parameter process.

Completely specifying a random process is considerably more difficult than specifying a random variable. Let a cdf, $F_X(x,t)$ be defined, for each allowed t, which is given by $F_X(x,t) = P[X(t) \le x]$. Further, for each of n allowable values of t, a joint cdf may be defined for the process, where $F_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n;t_1,t_2,...,t_n) = P[X(t_1) = x_1; X(t_2) = x_2;...; X(t_n) = x_n]$. Some properties of random processes may now be defined.

When X is continuous, $E[h(X)] = \int_{-\infty}^{+\infty} h(x)f(x)dx$. Note that, when two random variables X and Y are independent, expectation is a linear operator i.e. that $E[XY] = E[X]E[Y]$ and $E[X + Y] = E[X] + E[Y]$.

The variance of an RV X, defined on a discrete sample space, is given by the formula:

$$V[X] = \sigma^2 = \sum_i (x_i - E[X])^2 p(x_i)$$

If X is continuous, its variance is given by $V[X] = \sigma^2 = \int_{-\infty}^{+\infty} (x - E[X])^2 f(x)dx$. Note that $V[X] = E[X^2] - E^2[X]$ and that for independent RVs, variance is a linear operator, i.e. $V[\sum X_i] = \sum V[X_i]$.

Knowing the expected value and variance of a random variable, the **square of the variation coefficients** (svc) of the RV may be calculated, according to $Kx = V[X]/E^2[X]$.

3.2.3 Random Processes

A stochastic (random) process is a function of two arguments - time and a probability space, and is therefore denoted by the term $X(t,\omega)$. For a fixed value of t, $X(\cdot,\omega)$ is merely a function of the probability space Ω - i.e. is a random variable. For a fixed value of ω, $X(t,\cdot)$ is a function of time and is referred to as a ***sample function*** of the process. Examples of random processes include:

- The number of call requests that can arrive at a switch in [0,t) is a discrete-state, continuous-parameter random process.
- The waiting time of an inquiry for processing is a continuous-state, continuous-parameter process.
- If $\{X_n, n = 1,2,..,7\}$ denotes runtime of a job, where *n* is the day of the week on which the job is running, this is a continuous-state, discrete-parameter process.
- If $\{X_n, n = 1,2,...,365\}$ describes the number of jobs per day of the year, this is a discrete-state discrete-parameter process.

Completely specifying a random process is considerably more difficult than specifying a random variable. Let a cdf, $F_X(x,t)$ be defined, for each allowed *t*, which is given by $F_X(x,t) = P[X(t) \le x]$. Further, for each of *n* allowable values of *t*, a joint cdf may be defined for the process, where $F_{X_1,X_2,...,X_n}(x_1,x_2,...,x_n;t_1,t_2,...,t_n) = P[X(t_1) = x_1; X(t_2) = x_2;...; X(t_n) = x_n]$. Some properties of random processes may now be defined.

3.2.3.1 Properties of Random Processes

Independence: If $X(t_1)$ is independent to $X(t_2)$ etc. (say if X(t) is defined by the toss of a coin or the roll of a die), then $F(x_1, x_2, ..., x_n; t_1, t_2, ..., t_n) = F(x_1; t_1)F(x_2; t_2)...F(x_n; t_n)$.

Stationarity: In a stationary process, $F(x_1, x_2, ..., x_n; t_1 + h, t_2 + h, ..., t_n + h) = F(x_1, x_2, ..., x_n; t_1, t_2, ..., t_n)$. The distribution is independent of time over a set interval, i.e. the probability of changing from one state to another remains constant with time.

Wide-Sense Stationarity: Also referred to as Covariance Stationarity. The expected value and variance of the random process is independent of time, i.e.

$$E[X(t_i)] = \mu, \forall i \quad \text{and} \quad E[X(t)] = \mu, \forall t \in T$$
$$Var[X(t_i)] = \sigma^2, \forall i \quad \text{and} \quad Var[X(t)] = \sigma^2, \forall t \in T$$

Note: Stationary $\subset$ Wide-Sense Stationary

3.2.3.2 Some Common Random Processes

The Markov Process: A stochastic process is classified as a Markov Process iff

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

i.e. the future state of a Markov Process is dependent only on the current state, and not on past history. Markov processes have a number of noteworthy characteristics:

- If the state space of a Markov Process is discrete, it is referred to as a Markov Chain.
- In a Markov Chain, a state transition occurs at each discrete time unit, even if the state does not change as a result of the transition.
- If the chain is homogeneous or stationary, the future state is dependent only on the current state, and not on the time index associated with the current state.
- In a Markov Chain, the length of time spent in a state is defined by $P[\text{system remains in same state for } m \text{ transitions}] = (1 - p_{ii})p_{ii}^m$, i.e. it is geometrically distributed.
- In a Markov Process, the length of time spent in a state is exponentially distributed.

The Semi-Markov Process: The whole concept of Markov Processes revolves around the fact that a transition must be made at every unit time, even if the state remains the same after a transition. With Semi-Markov Processes, no such time restrictions are in place and the process may remain in a given state for a length of time defined by an arbitrary distribution. At transition times, the process behaves just like a Markov chain, and is referred to as an embedded Markov Chain.

The Birth-Death Process: A birth-death process is a type of Markov Chain (with geometrically distributed times between changes in state) and may be either a continuous- or discrete-parameter process, but has the added characteristic that state transitions may only take place between neighbouring states.

An example of a birth-death system is a queuing system where the time intervals approach zero (i.e. a continuous-parameter system), so that only one event can happen in an interval. Therefore, if a chain is in state $X_n = i$, the only possible events are:

An arrival $\Rightarrow X_{n+1} = i + 1$,

A departure $\Rightarrow X_{n+1} = i - 1$, or

Nothing $\Rightarrow X_{n+1} = i$.

The probability of a BDP being in a particular state k at time t is denoted by $P_k(t)$, where:

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

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

Using a State-Transition-Rate diagram, the rate of change of probability "flow" into state k equals the flow in minus the flow out. Therefore, the above equations may be restated in the form:

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

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

Let us define p_k as the probability that the system is in state E_k at some arbitrary time in the future, where $p_k = \lim_{t\to\infty} P_k(t)$. For the general BDP,

$$p_k = p_0 \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}} \qquad k = 0,1,2,\ldots$$

$$p_0 = \frac{1}{1 + \sum_{k=1}^{\infty} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_{i+1}}}$$

The Random Walk: This process is a form of Semi-Markov ***Chain*** in which:

$$X_{n+1} = X_n + U_n$$

$$\Rightarrow \text{for a random walk, } X_{n+1} = \sum_{i=0}^{n} U_i$$

where U_n is drawn independently from an arbitrary distribution

The Renewal Process: The Renewal Process is a specific application of the Random Walk, but is more specific in that it counts the ***number*** of state transitions of the monitored process. Therefore, the renewal process, evaluated at time t, will show the number of state transitions the subject process has undergone in the interval $[0,t)$, whether that interval is discrete or continuous.

Note that if the monitored process is a Markov Chain, with the state time geometrically distributed, or a Markov Process, with the state time exponentially distributed, the Renewal Process will also be Markovian, with the same state time distribution.

The Poisson Process: The Poisson Arrival Process may be defined by evaluating $P_k(t)$ for a BDP with zero departure rate and constant arrival rate λ. The result is:

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

The Poisson Process is also a specific form of the Renewal Process, in that it counts the number of state transitions in a specified time interval. Therefore, the Poisson process is a Renewal Process, a Random Walk, a Birth-Death Process (as only one state transition (or arrival) may take place at a time) and a continuous-parameter Markov Chain. Other important characteristics of a Poisson process include:

- For a Poisson Process, the average number of arrivals in $(0,t)$ is λt. Also, the variance of the number of arrivals in the same time interval is also equal to λt.
- For a Poisson Arrival Process, the interarrival times are exponentially distributed, i.e. the pdf of the interarrival times may be characterised according to $f(t) = \lambda e^{-\lambda t}, t \geq 0$. The mean of the exponential interarrival time distribution is $1/\lambda$, while its variance is $1/\lambda^2$. This distribution also has a property called **memorylessness**, whereby the distribution of the time until a future arrival is independent of the time since the last arrival, i.e. if, at some random time t, an estimate of the amount of time till the next arrival is evaluated, the result will be independent of the time that has elapsed since the last arrival.
- The Poisson process has stationary independent increments, i.e. events occurring in non-overlapping intervals of time are independent of each other.
- The Poisson process is covariance stationary (WSS) with:

$$E[X(t) - X(s)] = \lambda(t - s)$$
and
$$Var[X(t) - X(s)] = \lambda(t - s)$$

Figure 3.6 demonstrates the relationships between the above-described random processes.

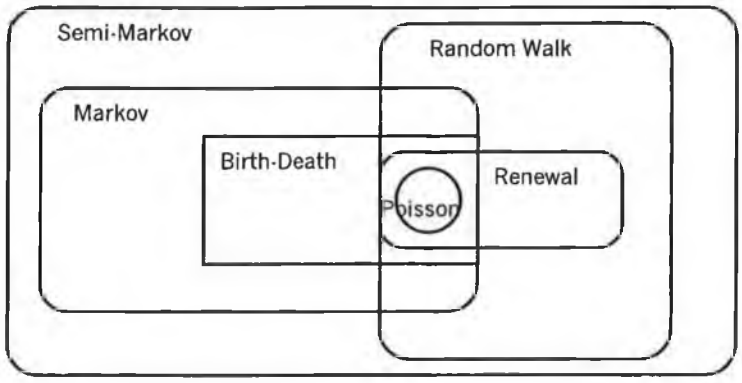


Fig. 3.6: Relationship between Random Processes

Note that the Poisson process has all the characteristics of all the other types of random process described above. It has also been found to be extremely suitable and accurate for modelling, amongst other things, the traffic in switched telephony networks. Therefore, in the model to be developed, all call arrivals to the IN SSPs will have a Poisson distribution.

3.2.4 Queuing Theory

A queuing system is any system in which arrivals place demands upon a finite-capacity resource [Kleinrock]. In particular, if the arrival times or the processing requirements of these demands are random, then conflicts for the use of the resource will arise and queues will form. A simple example is a queue for service at a cashier's desk in a bank. The length of the queue depends on:

- The average rate at which demands arrive. If the average arrival rate of customers is greater than the rate at which the cashier can serve customers, a queue will grow (and grow and grow) unless the cashier speeds up or another cashier provides assistance.
- The variation in rate at which demands arrive. If the cashier can handle the average rate at which customers arrive comfortably, a queue will still build up occasionally based on variations which will occur in arrivals. If these variations from the mean arrival rate are large, large queues will build up occasionally, whereas small variations will cause small queue build-ups.
- The average rate at which demands are served. Different customers will have different demands, and therefore the time to serve them will differ. If the average service time is shorter than the average interarrival time, the queue should not grow significantly.
- The variation in rate at which the demands are served. Even if the average service rate is greater than the average arrival rate, a queue will still build up occasionally based on variations which will occur in service times (e.g. a customer requiring a complicated transaction may require significantly more service time than the average). If these variations

from the mean service time are large, large queues will build up occasionally, whereas small variations will cause small queue build-ups to occur more often.

These dependencies apply to all queues, whether of customers in a bank or of call requests in a telecoms switch. Some common definitions used in queuing theory will now be provided.

- The arrival process to a queue may be described in terms of the probability distribution of the interarrival times of requests at the queue, *A(t)*, where $A(t) = P[\text{time between arrivals} \leq t]$. The mean arrival rate is denoted λ, with the mean interarrival time $1/\lambda$. The square of variation coefficient (svc) of *A(t)* is given by $Ka = V[A]\lambda^2$.
- The server process of a queue may be described in terms of the probability distribution of the service times of requests at the queue, *B(x)*, where $B(x) = P[\text{service time} \leq x]$. The mean service rate is denoted μ, with the mean service time $1/\mu$. The variability of *B(x)* may be assessed by the value of its square of variation coefficient (svc), given by $Ks = V[B]\mu^2$.
- The load of a queue, ρ, is effectively a measure of the proportion of time the processor of a queue is busy, and is calculated as $\rho = \lambda/\mu$. Note that, if $\rho < 1$ (i.e. $\lambda < \mu$) for a queue, the queue will be stable (i.e. will not become overloaded).
- Little's Law states that $\bar{N} = \lambda T$, where $\bar{N}$ is the mean number of requests in the system and T is the average time spent by requests in the system.

Queues can also be characterised according to the simple, 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 exponential (M), deterministic (D), erlangian (E) and general (G). So, for example, an M/D/2 queue is a two-server queue with exponential interarrival times (i.e. a Poisson Arrival Process) and deterministic (constant) service times.

3.2.4.1 The M/M/1 Queue

The M/M/1 queue is probably one of the simplest queues to analyse and is also very popular for use in modelling telecoms systems, as its behaviour is quite close to the mean behaviour of a real telecoms switch. As shown by its notation, the M/M/1 queue is a single-server queue with exponential arrival rates (forming a Poisson Arrival Process) and service times. Therefore, both λ and μ are constant (independent of both time and state of the arrival and server processes). From this may be derived a number of formula relating to the state of the queue:

- The average number of customers in the M/M/1 system is given by $\bar{N} = \sum_{k=0}^{\infty} k p_k = \frac{\rho}{1-\rho}$

- The variance of the number of customers in the system is $\sigma_N^2 = \sum_{k=0}^{\infty}(k-\bar{N})^2 p_k = \frac{\rho}{(1-\rho)^2}$
- The average time spent in the system is $T = \frac{\bar{N}}{\lambda} = \frac{1/\mu}{1-\rho}$
- The svc of interarrivals to the queue is $Ka = \frac{V[A]}{E^2[A]} = \frac{1/\lambda^2}{(1/\lambda)^2} = 1$. The svc of the service times, Ks, also equals 1.

Note that, according to Burke's theorem, the interdeparture times from an M/M/1 queue are also exponentially distributed. Therefore, these formulae are also valid when applied to a network of M/M/1 queues, but only if the network is feedforward (i.e. there is no feedback between any queues) and if all queues have only a single service rate. Therefore, if a queuing network exhibits feedback or contains queues with multiple service rates, some other method must be used to evaluate e.g. the mean queue length and mean waiting time for each queue in the network.

3.2.5 Choosing an Appropriate Technique for the Analysis of an IN Queuing Model

A number of queuing network analysis techniques are available for analysing the behaviour of a network (or chain) of queues. Each of these techniques places different requirements on the structure of the queuing network and provides different results. Therefore, in order to facilitate the correct choice of a technique suitable for analysing the behaviour of an IN under congestion, the requirements on an IN analytic model should first be specified.

Firstly, an open model of an Intelligent Network is needed, i.e. a model in which calls arrive from some external source outside the network and calls depart to some external sink outside the network. The model must also support multiple service types, each of which has different routes through the network and (potentially) different processing requirements at each node in the network. It should also be possible to model the behaviour of various congestion control strategies under various overload conditions with the resulting requirement that the throttling (blocking) of calls must be facilitated. To evaluate the efficiency of each congestion control strategy, the load and queue length at each node in the network must be able to be evaluated, as well as the average service delays for each service type (as described in Chapter 2). The mean arrival rates and loads can be calculated directly from the analytic model, but the queue lengths and service delay results must be provided by the analysis technique. It is acceptable to assume that external arrivals to the system are Poisson in nature, but it would be preferable for there to be no constraint on the nature of the service time distribution.

Having stated the requirements on the analytic model, it is now possible to examine the available queuing network analysis techniques, to establish which one best meets these requirements.

A number of queuing network analysis techniques exist (e.g. Jackson and BCMP networks) which allow the behaviour of all queues in the network to be calculated exactly, in terms of returning the joint probabilities of queue lengths [Gelenbe], but these techniques place stringent requirements on the network, including:

- All service times must be exponentially distributed,
- External arrivals must be Poisson,
- There may only be one class of customer, or if there is more than one, the service times of the queues must apply to all classes equally.
- All queues must have unlimited capacity.

Use of these techniques is not appropriate as the constraints on customer classes mean that IN traffic cannot be accurately modelled and also because the solutions provided by the analysis are not useful – e.g. they do not facilitate the estimation of service delays.

The alternative option is to use an approximation method. Approximation methods allow various characteristics of the queues within the network to be approximated accurately. Each of the techniques available for analysing networks of queues make a number of assumptions and allow particular characteristics to be estimated. Table 3.1 summarise the assumptions made by and the results provided by two different approximation methods.

Approximation Method	***Assumptions Made***	***Results Provided***
The Mean Value Method	For open networks: 1. Each queue FIFO 2. Exponentially distributed service times 3. Unique service rate at each queue	1. The mean queue length of each queue 2. Average service delay for each service
The Decomposition Method	1. Open network, 2. Each queue has single server 3. Each queue FIFO	1. The mean queue length of each queue 2. Average service delay for each service

Table 3.1: Summary of Two Approximation Methods

Both methods provide us with the desired results, namely the mean length of each queue and the average delay for each service type. However, the Mean Value method constrains all service types to having the same service rate at each queue. This is not desirable in our model, as IN calls and

non-IN calls have very different service rates. The decomposition method, on the other hand, does not have this constraint – in fact, all the constraints associated with the decomposition method are acceptable within our model. Other methods exist (e.g. the aggregation method and the isolation method) which may also be applicable, but these generally depend on the use of numerical analysis methods. The decomposition method is both simpler (as it involves only the solution of a series of linear equations) and suitable for describing an IN model, and was therefore chosen as the approximation method to be used within our analytic model.

3.2.6 The Decomposition Method for Queuing Network Approximation

The decomposition method is one of the most widely used approximation methods. It provides good solutions for all networks, whether or not they may be expressed in product form. It is based on analysing each queue in the network (as shown in Figure 3.7) separately, in order to express, for each queue j, the svc of interarrival times Ka_j and the svc of interdeparture times C_j in terms of Ka_k and C_k $(\forall k \neq j)$ of all other queues in the network. This analysis results in a set of linear equations, which may then be solved simultaneously to find the solution for each queue.

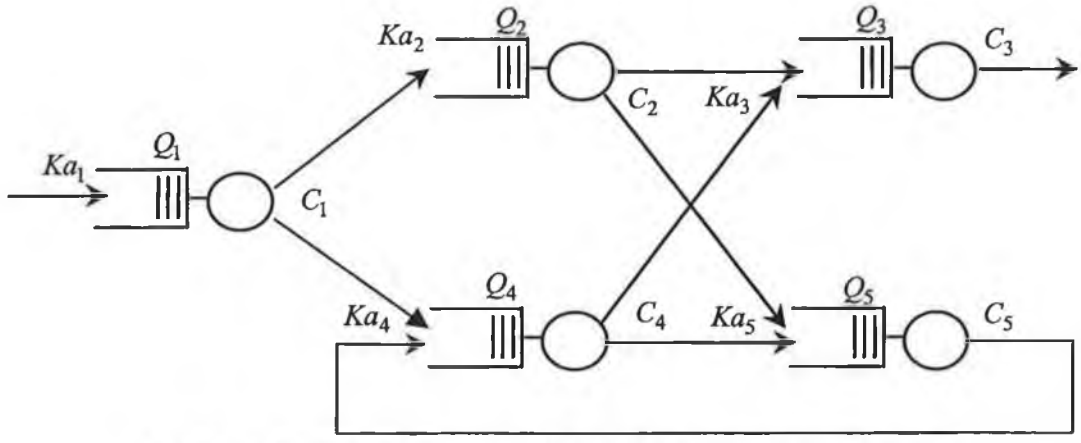


Fig. 3.7: Applying the Decomposition Method to an Example Queuing Network

The method is based on the assumption that the departure process from any station j is a renewal process (i.e. the time interval between two successive departures does not depend upon the preceding intervals). This assumption is valid in the case where the arrivals are Poisson and the service rates are distributed exponentially or when the station is saturated. As an intuitive justification of this assumption, the memoryless property of the exponential distribution has already been observed - thus independence of the interdepartures is consistent with this property. In a saturated server, there are always more customers than there are servers. This means the mean interval between successive departures from the station is the mean service time for the customer when all servers are busy.

3.2.6.1 Formulation of the Decomposition Method

The derivation of the equations for the decomposition method is provided in [Harrison] and will not be reproduced here. However, these equations and the various steps involved in the decomposition method may be outlined as follows:

Step 1: Evaluate C_j for each queue j in a network of K queues serving R different service types using the following equations:

$$C_j = -1 + (1-\rho_j)(Ka_j + 1 + 2\rho_j) + \lambda_j \sum_{r=1}^{R} \rho_{jr}\mu_{jr}^{-1}(Ks_{jr}+1) \quad \text{for } j = 1..K$$

where $Ka_j = \frac{1}{\lambda_j}\sum_{k=0}^{K}\left[(C_k - 1)P_{kj} + 1\right]\lambda_k P_{kj}$ for $j = 1..K$ and $P_{kj} = \sum_{r=1}^{R}\sum_{r'=1}^{R}\frac{\lambda_{kr}}{\lambda_k}p_{kr,jr'}$

The meanings of the terms used in the equations are: ρ_j is the load at queue j, λ_j is the total arrival rate at queue j, ρ_{jr} is the load at queue j due to call of service type r, λ_{kr} is the arrival rate of calls of service type r at queue k, μ_{jr} is the service rate at queue j for service type r, Ks_{jr} is the svc of the service rate at queue j for service type r (note that if queue j has an exponentially distributed service time for service type r, $Ks_{jr} = 1$) and P_{kj} is the total probability that calls departing station k move to station j, with $p_{kr,jr'}$ as the probability that a call of type r, on departing queue k will arrive at queue j as a call of type r'.

The result of this step is a series of linear equations (one for each queue in the network), with C_j expressed as a function of the svc of the service time at station j and the variations in the arrivals from the other stations in the network (which in turn depend on the svc of the departures from each of these stations).

Step 2: Solve the series of linear equations C_j, $1 \le j \le K$ simultaneously. The result may then be used to evaluate Ka_j, $1 \le j \le K$, for each queue in the network.

Step 3: The mean length of the j^{th} queue $\overline{L_j}$ can then be approximated for each station by Kingman's formula (a generalisation of the Pollaczek-Khintchine formula):

$$\overline{L_j} = \rho_j\left(1 + \frac{\rho_j(Ka_j + Ks_j)}{2(1-\rho_j)}\right)$$

Step 4: The mean delay at the j^{th} queue $\overline{T_j}$ may then be found through Little's law $\overline{T_j} = \overline{L_j} / \lambda_j$, and the mean delay for each service type can be evaluated by summing the delays over all queues in the route taken by the service type.

In summary, the decomposition method studies each queue in the network on a queue by queue decomposition of the network. In this method, the distributions of the service times and interarrival times at each station j are approximated by the total mean rate of arrivals λ_j and the square of their variation coefficients (i.e. the variation multiplied by the rate squared). In this way, the arrival streams from other station in the network and from the exterior can be approximated. The mean queue length is then found using Kingman's formula and the mean delay using Little's Law.

3.3 Mathematical Optimisation

Mathematical optimisation describes a methodology for evaluating, where it exists, the maximum or minimum value of a mathematical function (and the values of the function parameters which provide this optimised solution) subject to a number of (equality or inequality) constraints. The mathematical function to be optimised is referred to as the **objective function,** *Z*. The possible values of the objective function, subject to the constraints, form a **feasible region** (an area in which all points are possible solutions to the objective function and obey all constraints). Optimisation of the objective function returns the highest (or lowest) point in the feasible region. Note that if an optimisation problem is not carefully defined, one of two considerable problems may result:

- **An infeasible solution**: If there is no feasible region in which all constraints are satisfied, it will not be possible to generate an optimum solution, and the algorithm will deadlock.
- **An unbounded solution**: If the feasible space is unbounded (i.e. is not constrained on all sides), the optimisation may never complete, as the algorithm may find on each iteration, a result that is greater than (or less than) the result found on the previous iteration. This will effectively result in a livelock of the optimisation algorithm.

An optimisation problem may be linear or non-linear. Graphical examples [Greenberg] of a two-dimensional (two parameter) linear and non-linear optimisation problem are provided in Figure 3.8.

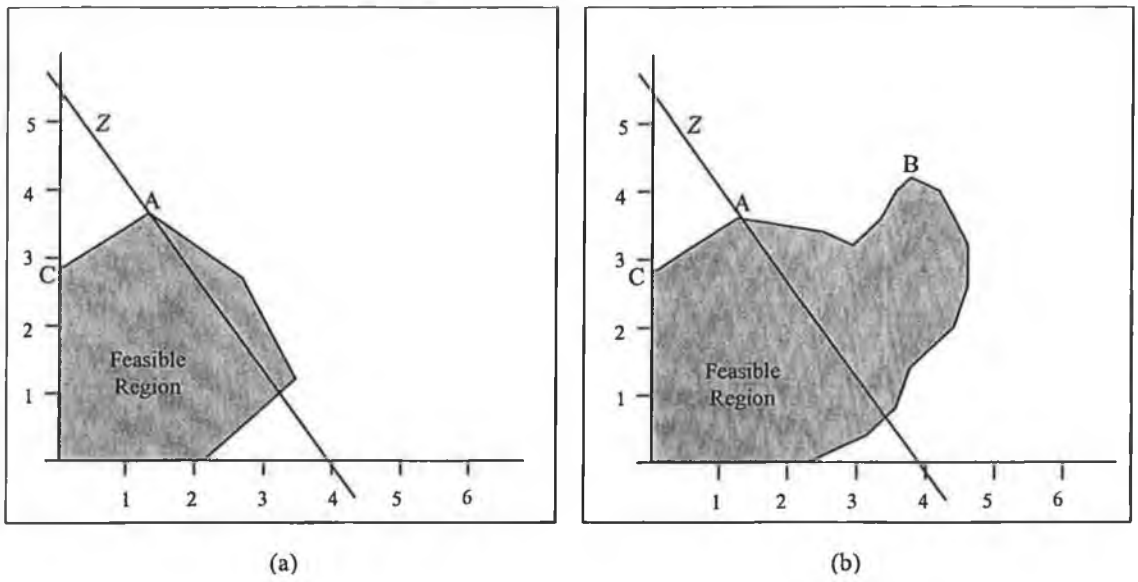


Fig. 3.8: (a) Linear constraints, (b) Non-linear constraints

Note that, if the search for the optimum solution begins at C and progresses along the bound of the feasible space (along the line segment [AC], which represents the points for which the constraint defining the line AC is **active**, i.e. the left hand side of the inequality equals the right hand side), the linear search will find A, which is the correct solution for the problem. However, for the non-linear problem, there is a risk that A will be selected as the optimal solution, as it is a local maximum. This would be incorrect, as the actual global optimum is point B. This demonstrates (intuitively) that non-linear optimisation is more complicated than linear optimisation. If it is desired to use an optimisation technique for congestion control, it must fulfil one of the basic requirements on a congestion control algorithm, which is that the algorithm must be simple and efficient. It is therefore preferable that a linear optimisation algorithm should be used for congestion control. In order for an objective function and corresponding constraints to form a valid **Linear Programming Problem** (LPP), there are two requirements that must be adhered to. The first is that all parameters must be non-negative. This is not a problem for a congestion control algorithm, as all parameters and constraints will relate to loads, probabilities and priorities, which can only have non-negative values. The second requirement is (obviously) that, the objective function and constraints must be kept linear. This does constrain the algorithm somewhat. However, if the objective function and constraints are expressed intelligently, the benefits of keeping the problem linear outweigh the overheads that would be required to express a more comprehensive problem in non-linear terms.

The most commonly used method for solving linear programming problems is the Simplex Method. There are two forms of this algorithm – the single-phase and two-phase simplex algorithms. The only difference between the two forms is how the initial search point is

established, and this is dependent on whether each constraint has an upper bound (i.e. that the function of the parameters is less than some constant). The single-phase form requires that all constraints are upper-bounded, while the two-phase simplex algorithm can handle constraints for which only lower bounds are defined. For a congestion control strategy, the requirements will always be that some value (load, number of calls, etc.) is not exceeded, and therefore all constraints will be upper-bounded. Therefore, the single-phase form of the simplex method is suitable for congestion control algorithm specification.

The software used to implement the simplex method was LP_SOLVE [Berkelaar95] and was acquired as freeware from the Internet.

Chapter 4

Comparison between Existing SCP Congestion Control Strategies

In this chapter, we compare the behaviour and efficiency of the most common strategies used in IN congestion control. As Chapter 2 describes, a number of publications, using various types of extremely simplified IN models, present conflicting results about the benefit of these strategies (e.g. [Pham92], [Tsolas92]), and so none of the results presented may be accepted as being conclusive. Therefore, a comprehensive IN simulation model was developed to compare these strategies, with the aims of:

- Establishing which of the existing strategies performs best under various load conditions, and
- Identifying any problems and difficulties associated with both developing and executing these strategies,

in order both to clarify the requirements on a new IN congestion control strategy and to suggest which type of algorithm may prove useful as the basis for this new strategy.

Note that the emphasis of this work is on the efficient protection of the SCP, as being the physical entity whose behaviour is most critical to IN performance. In Section 4.1, the IN simulation model which was developed to facilitate this investigation is described. In Section 4.2, the method by which each of the various congestion control strategies were implemented on the model is outlined, while Section 4.3 compares the operation and effectiveness of the strategies to establish which is the best of the existing strategies. Finally, Section 4.4 summarises the results of simulations and highlights the drawbacks of the existing strategies.

4.1 The IN Simulation Model

The best way to approach describing the simulation model is to initially describe how it was designed, including any simplifications made to the standard IN concepts. Each simplification will be justified. Also at this point, the services which were designed to execute on the model will be described, in terms of the actions and information flows required in the real world, and therefore in the model, to execute services of this type. These descriptions and justifications will be provided in Section 4.1.1. Once the model design has been described, the details of how it was developed at each level of the OPNET package will be described in Sections 4.1.2 and 4.1.3.

4.1.1 Overview of the model

The IN model was designed to be able to handle both IN and non-IN calls (see Figure 4.1). It consists of two SSPs that are fully connected to each other, thereby permitting the exchange of control messages required to setup and teardown an ISDN call. Each SSP contains a Call Control Function, which is responsible for completely executing non-IN calls and for detecting any service-related requests in new calls. Any service requests detected in the CCF are forwarded to

the Service Switching Function, which is responsible for communication with the SCP. So far, the operation of the SSP is as specified in the standards. However, it was decided to simplify the model by integrating the Service Resource Function of the Intelligent Peripheral into the SSP. This is justifiable as every IP request begins with a request from the SCP to the relevant SSP to open a channel between the IP and the user, before instructions are sent to the IP and interaction with the user occurs. Therefore, giving the SSP control over SRF execution simplifies the model without affecting the sequence of events involved in service processing. A second assumption is also made regarding the SSPs and relates to their capacity – as the aim of this work in this Chapter is to compare commonly used SCP congestion control strategies, the SSP capacity is defined to be very large so that the SSP will never become congested and influence the results for the strategies. This assumption will be removed in Chapter 5, where the effects of SSP overload on IN performance are investigated.

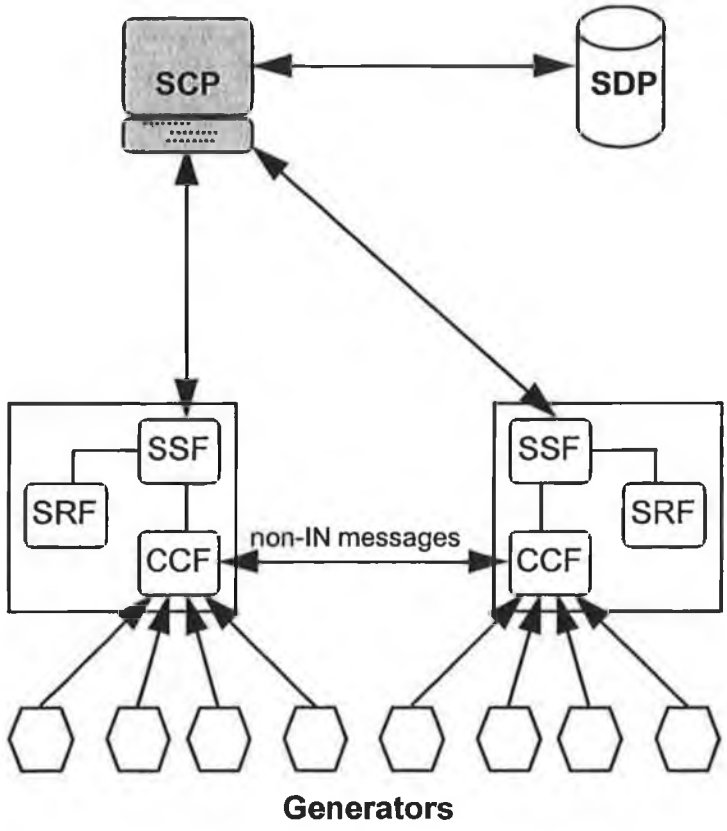


Fig. 4.1: The Simulation Model

Another simplification in the model involves the connection of each SSP directly to the SCP - the operation of the SS7 network that, in reality, would be responsible for handling communications between these physical elements is not addressed. This is based on the assumption that the dimensioning of the SCP was done intelligently with respect to the capacity of the SS7 links, so that (as the SS7 is dimensioned so that it generally runs at approximately 0.4 Erlangs under normal traffic loads while SCPs are generally dimensioned to run at about 0.7 Erlangs) in almost all cases,

SCP overload will occur prior to SS7 link overload [Lodge98b]. In this case, the only impact of the SS7 network is the delays that call requests exhibit in transit between IN physical elements. As modelling delays in the SS7 is a very complex task and does not add much insight into this investigation, it was decided to omit the SS7 from the model. Therefore, the SSPs and SCP are connected directly to each other, as shown in Figure 4.1, and transmission delays are assumed to be negligible. The protocol underlying the connection between the SCP and the SDP is not explicitly defined in CS-1, and therefore it is assumed, for simplicity, that there is a dedicated line between the control point and the service database with negligible transmission delays (this would be exactly the case if, for example, the database was an integral part of the SCP).

This model is very similar to that developed by [Leever93] in his estimation of IN service performance and is very different to that used in most research studies to date, as it does not simplify the operation of the IN into just a simple exchange of messages between SSP queues and SCP queues, but also integrates interactions with the SDP and IP. Therefore, the actual characteristics of service traffic between PEs of an IN may be represented, along with the added advantage that no assumptions are made as to the delay involved when requests are sent to the SDP and IP. In previous research, the delays involved in accessing data or interacting with users, if modelled at all, was modelled as either a constant or distributed delay between SCP processing times. The fact that we do not make this assumption means that traffic behaviour and associated statistics gained from the model simulation should more correctly approximate the load profile of each service type in the system.

In order to apply the strengths of the model architecture, it is necessary to reproduce, as accurately as possible, the various information flows and processor requirements that would occur in a real network offering real services. Information relating to real service processing requirements is not made public by network operators or equipment vendors, and so, unfortunately, it is only possible to attempt to estimate it intelligently. Regarding service offerings, we chose to implement two IN services - Televoting and Freephone – as well as the non-IN (ISDN voice call) service. Note that, as a simplification to the realisation of these services, all call requests which are not rejected in the SSPs are assumed to complete correctly – i.e. non-IN and freephone calls will be setup and routed correctly and the terminating party will always accept the calls, while it is assumed that all televoting callers interacts correctly with the service and register their votes correctly.

The DFP IFs between PEs and the relevant processor actions at each PE required to complete IN service requests of each type is described below, as is the realisation of the non-IN calls. A written description of each operation is also provided for each call type.

4.1.1.1 The Televoting Service

The televoting service involves the collection of data from users who may be located anywhere in the network and is primarily used when general opinions of a population are desired. Usually, the number of the televoting service is broadcast over one or more types of media (e.g. television, radio, newspapers etc.) and many callers ring the number to volunteer an opinion on a topic. When a user calls, they are connected to a voice messaging system, which prompts them to enter their choice by pressing some sequence of digits. When the choice is entered, it is used to update the information in a database, and the users choice is acknowledged. The call then terminates. The CS-1 information flows may be represented as shown in Figure 4.2 below. In the diagram, the arrowed lines depict information flows, with the name of the IF above the line and the information carried in the message in italics below the line.

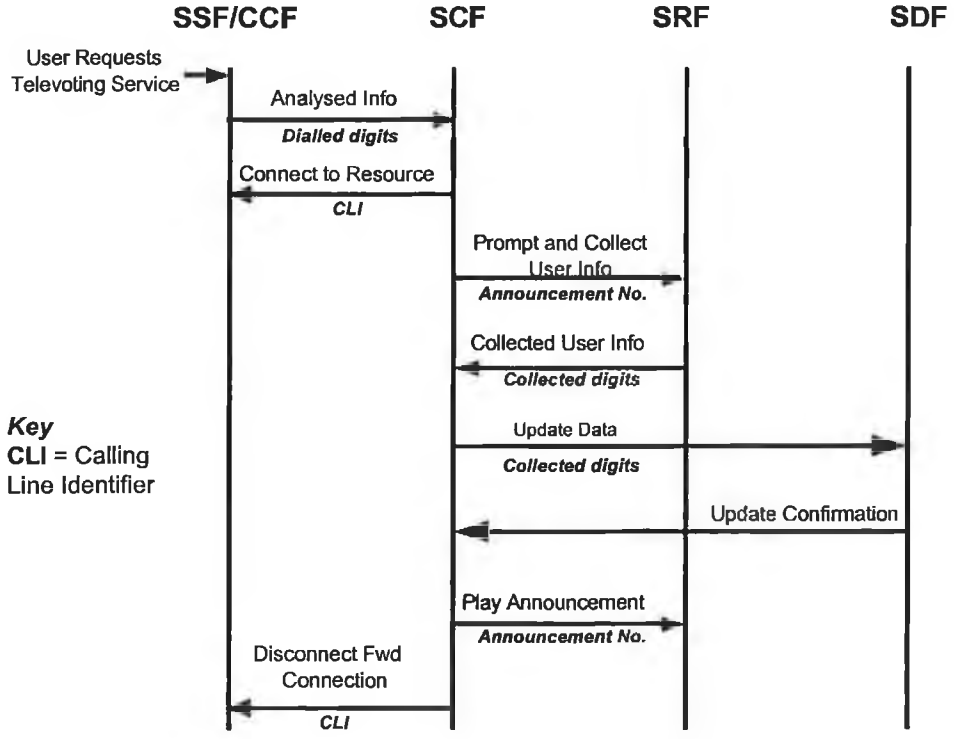


Fig. 4.2: Decomposition of Televoting Service

The series of actions required to execute an instance of the televoting service may be described as follows:

- User goes offhook and dials the televoting number,
- The digits typed in by the user are collected and examined by the CCF,
- A service request is recognised by the CCF and call processing is suspended,
- The SSF builds an Analysed_Info message containing the dialled digits and sends it to the SCF,
- The SCF creates a new instance of the Televoting Service Logic Program (SLP),

- The SLP Instance (SLPI) sends a message to the SSF requesting that a channel be opened between the user and the SRF,
- The SLPI sends a message to the SRF requesting that the relevant announcement be read and the digits keyed in by the user in response be collected,
- The SRF collects the resulting digits and passes them back to the SCF,
- The SCF sends a message to the SDF in order to update the televoting statistics,
- The SDF sends an acknowledgement,
- The SCF requests the SRF to play an acknowledgement announcement to the user,
- The SCF tells the SSF to disconnect the SRF from the user and to end the call. The SLPI then terminates.
- The SSF terminates call processing.

4.1.1.2 The Freephone Service

When a customer subscribes to the freephone service, they are allocated a number (in Ireland the number allocated is '1800' + 6 digits), which is their freephone reference number and not related to their actual destination number. When a user dials this number, it is interpreted by the SCP and, by accessing a database, the associated destination number is acquired. This number is returned to the SSP for call routing and connection. When the conversation terminates, the subscriber is charged for the call. The call setup procedure is shown in Figure 4.3 below - note that the charging aspect of the call is not addressed here.

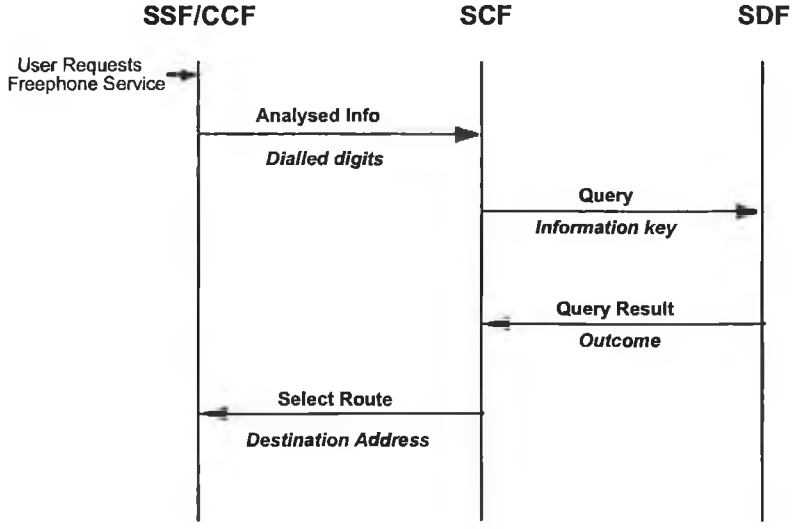


Fig. 4.3: Decomposition of Freephone Service

The information flows may be interpreted as follows:

- The user goes offhook and dials the freephone number,
- A service request is recognised by the CCF and call processing is suspended,

- The SSF builds an Analysed_Info message containing the dialled digits and sends it to the SCF,
- The SCF creates a new instance of the Freephone SLP,
- The SCF sends a message to the SDF requested the appropriate destination number,
- The SDF returns the number,
- The SCF tells the SSF to connect the user to this number,
- The SSF continues call processing with the correct destination number (i.e. it routes the call in the same way as it would a non-IN call).

4.1.1.3 Realisation of non-IN calls

The implementation of non-IN calls was realised using an approximation of the ISDN control message sent between the SSPs. As shown in Figure 4.4 below, when the CCF of an SSP receives a non-IN request, it generates a termination request and transmits it to the other SSP in the network. When the termination request arrives, it causes the number of active channels between SSPs to be incremented and returns an acknowledgement message to the originator. (Note that the model was developed so that if it was desired to limit the number of channels between switches, an upper bound could be set on the channel number and if this was reached, the call request would be refused using a NOACK message). On receipt of an ACK message, the originator then sets a timer to simulate the length of the conversation - this length is derived from a uniform distribution of between 150 and 210 seconds. When the conversation finishes (the timer expires), a message is sent to the terminating SSP to decrement the number of active channels. In this manner, the control messages required to carry out a call are modelled without having to maintain the actual channels between the SSPs, while retaining the ability to monitor the number of active channels and use this as a criterion for accepting non-IN calls.

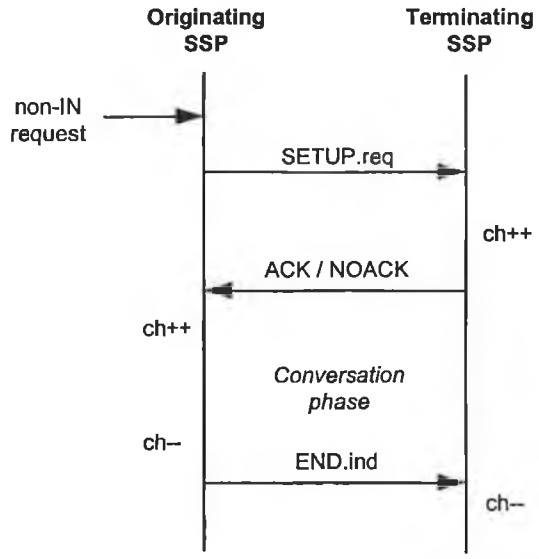


Fig. 4.4: Non-IN call handling

4.1.2 The Network Layer Model

The network layer model in OPNET is depicted in Figure 4.5. As shown, the model consisted of two SSPs fully connected to each other to permit the accurate representation of the control messages required to setup and teardown a non-IN call - note that each link between SSPs is defined for one of the control message types outlined in Section 4.1.1.3. There is one SCP, which is directly connected, via two unidirectional links, to each SSP and to the SDP.

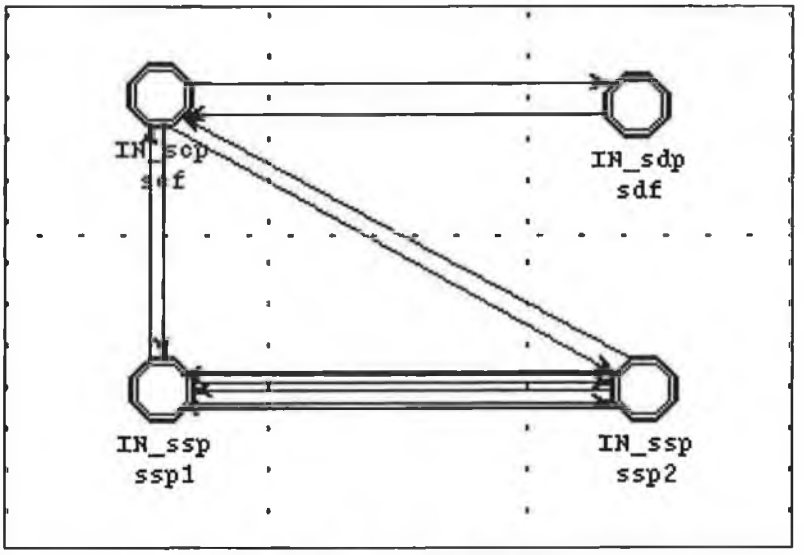


Fig. 4.5: The Network Model

4.1.3 The Node Layer Models

As shown in Figure 4.5, the network model consists of four interconnected nodes. These are realised using three node models, namely, the **IN_scp**, the **IN_ssp** (ssp1 and ssp2 in the network model are instances of this model) and the **IN_sdp**. The structure of each node will now be described, in terms of the process models that form them and the information streams between these processes.

4.1.3.1 The IN_ssp Model

The operation of the SSP (and IP) is realised by the interactions between a number of different queuing models, as shown in Figure 4.6.

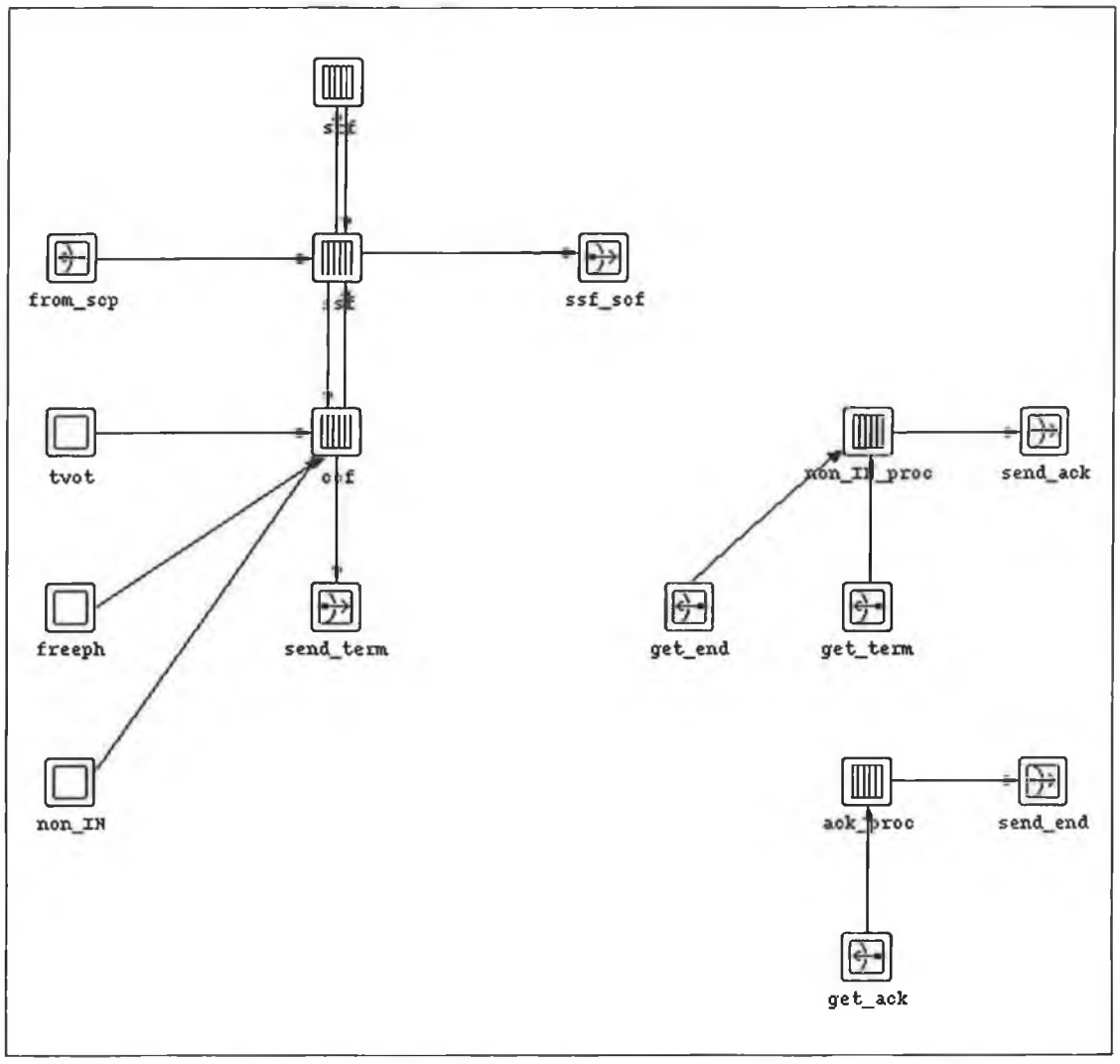


Fig. 4.6: The IN_ssp node model

Input requests are generated as packets by instances of the *IN_gen* process. Three instances of this generator exist within the node, each one being responsible for the generation of one of the call request types - namely Televoting, Freephone and non-IN calls. Each generator produces packets according to a Poisson distribution, with a mean arrival rate that may be re-specified at the start of each monitoring interval (this allows the arrival rates for each traffic type at each SSP to be varied independently over the course of a simulation). The generated packets are of a specified format, being of length 10 bits and containing the fields:

- **Service:** This field contains an integer that defines the type of service being requested. Each generator instance sets this field accordingly (i.e. non-IN = 0, freephone = 1 and televoting = 2).
- **Msg_type:** This field contains integers that may be set/updated by any PE to denote which stage of call processing a particular request is in. For example, if the SCP wishes to access the IP, it will set this field to "PLAY_ANNOUNCEMENT", and transmit it to the SSP.

- **ssp_id:** This value is set when the packet is created in order to identify which SSP it was created in. This will be used by the SCP for addressing purposes.
- **Delay:** This field contains a float value that is initially zero. As the simulation proceeds, this value will be updated in order to dynamically track the delays experienced by this packet during processing at each PE along its route.
- **Param:** This field was included to cater for services for which the same function must be carried out more than once. For example, if a particular service requires database lookup on two separate occasions, on the first occasion Msg_type will be set to "UPDATE_DATA" with Param = 1, while on the second occasion, Msg_type will again be set to "UPDATE_DATA", but with Param = 2.

All new requests are routed to the *ccf* queue, where processing occurs in a First-In-First-Out (FIFO) manner to establish whether the call is a service request or a non-IN request. All service requests are forwarded to the *ssf* queue, while non-IN call setup requests are immediately transmitted to the other SSP via the *send_term* transmitter. The *ccf* queue is of infinite length with its mean (exponential) service rate promoted, so that it may be defined at runtime.

When new call requests arrive at the *ssf* from the *ccf* or the *srf*, they are transmitted immediately to the SCP via the *ssf_ccf* transmitter. When a message is received from the SCP, the *ssf* checks the contents of the 'Msg_type' field to establish whether it should be sent to the *ccf* or the *srf*, and forwards it to the correct process without delay.

The *srf* is modelled as an Erlang-C queue, with each of 20 virtual servers having a uniformly distributed service rate of +/-2 seconds around a mean of 5 seconds. This mimics the behaviour of an IP with 20 recorder devices, where the service time distribution represents the time when announcements are being read to the user and digits collected. After the service time has elapsed, packets are returned to the *ssf*, which sends them back to the SCP via the *ssf_scf* transmitter.

When a termination request arrives at the SSP, it is processed, without delay, at the *non_IN_proc* queue, the number of active channels is incremented and an ACK returned to the originator. The *ack_proc* processor receives the ACK message, and sets a timer for a uniformly distributed conversation time around a mean of 180 seconds, before transmitting an END message, which enters the *non_IN_proc* queue of the originator and causes the number of active channels to be decremented before destroying the packet.

When a packet associated with the freephone service has completed processing at the SCP, it is returned to the *ssf*, which forwards it to the *ccf*. This packet is processed by the *ccf*, and causes a non-IN call to be established with higher priority than new calls (because more resources have

been expended processing these calls than has been expended on new calls and therefore, to maximise efficient processor usage, it is imperative that these calls complete successfully).

Finally, messages relating to SCP overload arrive at the *ssf*, which forwards them to the *ccf*. All throttling mechanisms for the control of IN traffic (e.g. CG, Percent Thinning (PT) or Window) are located at the *ccf*.

4.1.3.2 The IN_scp Model

As shown in Figure 4.7, the functionality of the SCP is divided among a number of queues. All incoming requests arrive at the *scf_q* queue module, where their further processing needs are established. Any messages arriving from the SDP or SSPs must queue for service at the *scf_q* module. They are served in a FIFO manner with an exponentially distributed service rate (not explicitly defined within OPNET, so that it can be defined at runtime and therefore varied over a number of simulations). After the expiry of the service time, the packets are evaluated to establish which service they are connected with and are then forwarded to the relevant service *slp*.

Each *slp* is a simple FIFO queue with an exponentially distributed service rate and contains the knowledge of the routing information for the service is stored. When a packet arrives at an *slp*, the information contained in the 'Msg_type' and 'Param' fields of the packet provides details as to the last location at which the packet was processed. The route logic is then used to determine the next destination for the packet, and this information is embedded in the packet before the packet is sent to the *out_q* process. In *out_q*, the 'Sspid' and 'Msg_type' fields of the packet are read to establish where the packet is to be sent. The packet is then dispatched to the relevant transmitter without delay (*to_sdp*, *to_ssp1* or *to_ssp2*).

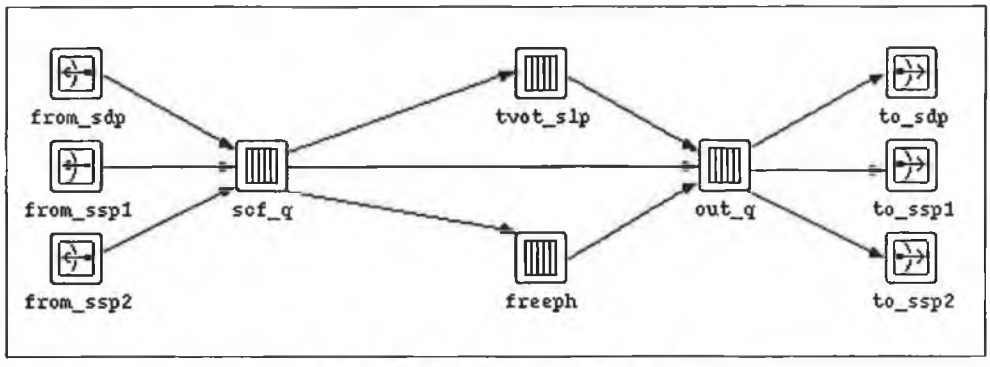


Fig. 4.7: The IN_scp node model

The *scf_q* module represents the central processor of the SCP. The *out_q* module of the model was only developed in order to distribute the functionality of the SCP among different elements of the model, thus ensuring that the code in the central processing module (*scf_q*) does not become overly complex.

4.1.3.3 The IN_sdp Model

The SDP model contains simply one receiver, one active queue and one transmitter as demonstrated in Figure 4.8. All arriving packets are from the SCP, and are inserted in the *sdp* queuing module. Here, the 'Msg_type' field of the packet at the head of the queue is analysed to discover whether a read or an update operation is required. The queue serves packets in a FIFO manner with two constant service rates defined (one for read operations, the other for update operations) - the assumption is made that an update operation takes longer than a read operation, so for the purposes of the simulation, the service time for an update operation is twice that for a read operation. After the service time has elapsed, the data in the packet is amended to show that it has received SDP processing and is returned to the SCP via the *to_scp* transmitter. Note that the service rates of the SDP were defined (relative to the SCP service rate) so that the SCP will become overloaded prior to the SDP.

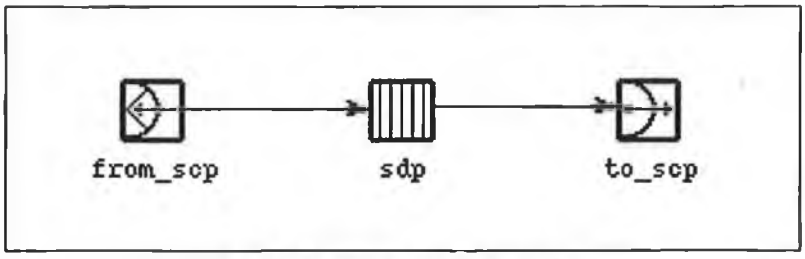


Fig. 4.8: The IN_sdp node model

4.1.4 Congestion Strategy Evaluation Criteria

The criteria used to evaluate the performance of the developed congestion control strategies are (as stated in Chapter 2, Section 2.3.5):

- User delays,
- SCP queue length,
- SCP load.

As the assumption is made during this investigation that SSP resources are infinite, SSP load is not measured. The efficiency of the algorithms is also estimated, in terms of their resource requirements, simply by comparing the duration of the simulations when all factors other than the congestion control strategies used (e.g. service rates and arrival rates) are identical. The methods used to acquire data to allow the quality of the control strategies to be calculated according to the above criteria will now be presented.

4.1.4.1 Measurement of User Delays

User delays for each service type were measured by:

- summing the delays (both queuing delays and processing delays) for each request at each node along its route through the network in order to find the total delay for each request (the delay experienced by each request is stored in the 'delay' field of its packet and is updated in each PE) and,
- calculating the average delay over a monitoring interval for requests for each service type and writing each result to a statistic within the *out_q* process of the SCP node.

However, the service time at the *srf* represents the time in service execution when the network is interacting with the user - i.e. the user is receiving a response from the network during this time, even though service execution has not yet completed. Therefore, the service delay experienced by packets there may be discounted. Note, therefore, when examining the delays for each service in Section 4.5.1, that in services which require user interaction, the delays may appear quite high (to the order of 1 - 3 seconds) this is due to the fact that information exchanges are required between the SCP and IP, and the user has received a response from the network during that period.

4.1.4.2 Measurement of SCP Queue Length

A probe is placed on the *in_q* process of the **IN_scp** node. This probe dynamically records the queue length over the course of a simulation and presents both the actual and the time-averaged value of the queue length as a result of the simulation.

4.1.4.3 Measurement of SCP Load

The SCP throughput for each control strategy was estimated by evaluating, at the end of a monitoring interval, the number of calls processed during the interval as a percentage of the total number of calls that could be handled by the SCP at full capacity. Note that, as the SCP does not reject any calls, SCP load and throughput are equivalent.

4.2 Implementation of Congestion Control Strategies

Two classes of congestion control strategies were implemented in the model. The active strategy, Window was implemented in the *ccf* (see Section 4.2.3). For the reactive strategies, four detection methods and two throttles were implemented. The detection methods, which were implemented at the SCP of the model, were:

- Call Count Control (CCC),
- Queue Length Control (QLC),
- Load Measure Control (LMC),

- Response Time Control (RTC) - this is a variation of a detection method used in mobile networks, and was originally developed by [Gulyani93] and then enhanced during the course of this work (see [Lodge94]).

Each of these detection methods was integrated into a different part of the SCP model. Reasons for the locations of the algorithms and a description of how they operate will be provided in Section 4.2.1. In all cases, fifteen different levels of overload were defined.

The two throttles that were implemented are CG and PT. Both of these were integrated into the *ccf* process of the **IN_ssp** model. Their implementation is described in Section 4.2.2.

Note that there are no controls in the model to protect against SSP overload. It is assumed for this work that the trunk capacity between switches is infinite, and that SSP *ccf* call processing capacity, while not infinite, is very high. Therefore, it was deemed unnecessary to protect the SSP from possible congestion. Also, it was decided to simplify the initial investigation of the behaviour of the strategies by evaluating their behaviour under load from one service type only. Therefore, all overload parameters associated with the various strategies were estimated based on the assumption that all calls are of the Freephone service type. This assumption will be removed later.

4.2.1 Implementation of SCP Congestion Detection Methods

4.2.1.1 Call Count Control

As described in Chapter 2, the CCC detection method must be implemented at the input queue of the processor that requires protection. As the resource that must be maximised in the IN is the processing capacity of the SCP, modelled as the *in_q* process of the **IN_scp** node, the algorithm must be placed at the input of this module. For this algorithm, an interrupt is generated automatically at the end of a pre-defined monitoring interval, resulting in a transition to a *monitor* state. When this state is entered, the number of ***new*** arrivals (note that returning messages from the SSP, IP and SDP were not included in this figure) within the previous monitoring interval is counted and compared with a pre-defined table of arrival/overload level values to find the current overload level. This result is used to derive the associated overload level using the algorithm described in Figure 4.9.

```
if current_level >= previous_level
      then new_level = current_level;
if current_level < previous_level
      then new_level = previous_level - 1;
```

Fig. 4.9: Algorithm for Estimating Overload Level

Note that this algorithm bases its calculation not just on the currently detected overload level, but also on the overload level that was in place during the previous monitoring interval. This is because taking the previous overload level into account reduces oscillations in the SCP load, as shown in Figure 4.10, where a CCC algorithm which uses the memory of the previous overload level to define the new level is compared to an algorithm where the new overload level is based purely on the number of arrivals in the previous interval (i.e. an algorithm with no memory). These reductions in oscillations result in lower average queue lengths (Figure 4.11) and therefore lower overall service delays (Figure 4.12).

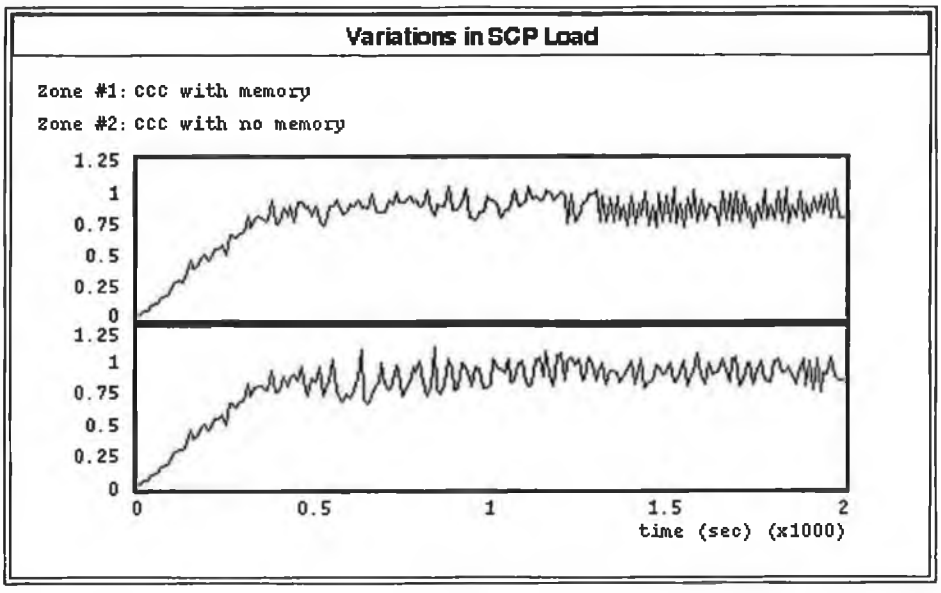


Fig. 4.10: SCP Load for CCC algorithm with Memory vs without Memory

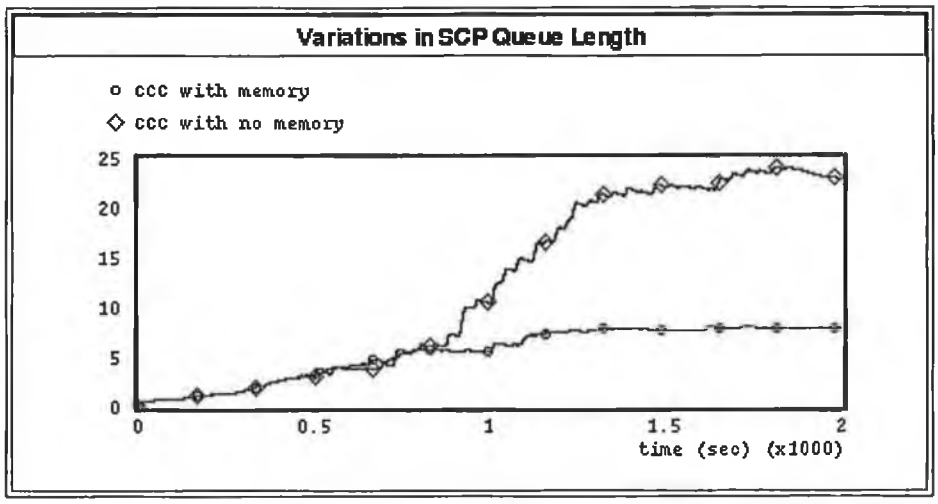


Fig. 4.11: SCP Queue Length for CCC algorithm with Memory vs without Memory

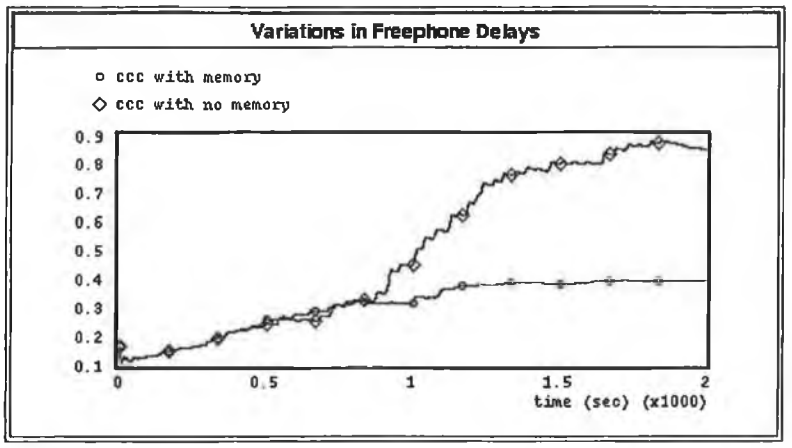


Fig. 4.12: Freephone Delays for CCC algorithm with Memory vs without Memory

If the current overload level is evaluated as greater than 0, an overload message (containing the current overload level) is created and sent, without delay, to both SSPs. Similarly, if the level evaluated is 0 when the previous level was greater than 0, an overload message is created to notify the SSPs that the overload condition no longer exists and that the throttle may be disabled.

4.2.1.2 Queue Length Control

The QLC algorithm was also implemented in the *in_q* process in the SCP node. For this, every time a new packet arrives and is inserted in the queue, the number of packets in the queue is established and compared against a list of queue lengths corresponding to the fifteen overload levels. The resultant level is then compared with the previous overload level, and the new overload level is determined using the algorithm described in Figure 4.9. The result is then encoded into a message and sent to the SSP where the throttle is invoked if required.

4.2.1.3 Load Measure Control

The LMC algorithm was also implemented in the *in_q* process in the SCP node and operates in a similar fashion as the CCC algorithm. An interrupt is generated automatically at the end of a pre-defined monitoring interval, resulting in a transition to a *monitor* state. When this state is entered, the mean load during the previous monitoring interval is evaluated and used to derive the associated current overload level from a pre-defined table of loads/overload levels. The resultant level is then compared with the previous overload level, and the new overload level is determined using the algorithm described in Figure 4.9. The new overload level is then encoded into a message and sent to the SSP where the throttle is invoked if required.

4.2.1.4 Response Time Control

The RTC algorithm is similar to CCC but is implemented at the *out_q* module of the SCP. The average service execution time for requests is measured over a constant monitoring period, by

causing a transition to state *monitor*. *Monitor* contains the code required to evaluate the average total delay of packets of each service type (in this case by using the 'delay' field of each packet, but in a real system, this could be accomplished by tracking the average duration of the lifetime of each SLPI). This delay is made up all the individual delays experienced by packets at the SCP, IP, and SDP. Therefore, this detection method has an advantage over CCC and QLC in that if the IP or SDP becomes overloaded, this will be noted in the *out_q* of the SCP, as the total delays of requests will be affected by the delays at these PEs. On the other hand, RTC is also more complex than CCC or QLC, as in the case when IN traffic is made up of multiple service types, the response time for each service type varies according to the requirements which the service places on the system, and therefore the average delay would have to be measured separately for each service request type. However, as for this investigation, traffic in the system is made up of only freephone calls, the implementation of this algorithm is made up of only freephone delays.

As RTC is implemented, at the end of a monitoring interval, the average response time of freephone calls during that interval is compared to the list of times corresponding to the fifteen overload levels and the resultant overload level for each service is determined using the algorithm outlined in Figure 4.9. The average overload level, if different from the previous overload level, causes notification is sent to the SSP in order to ensure that it restricts traffic according to the new overload level.

Note that, for all the detection methods described above, a table defining the correspondence between measured values and overload levels must be derived. These tables were derived through trial and error, which proved to be an extremely non-trivial task. Also, as the efficiency of any detection method based on the use of a monitoring interval is dependent on the length of that monitoring interval (as described in Chapter 2, Section 2.2.3.2), the behaviour of CCC, LMC and RTC was evaluated under various monitoring interval lengths over the course of a number of simulations, with the result that the use of a monitoring interval of 10 seconds was established as providing consistently the best results.

4.2.2 Implementation of Throttles

The CG throttle was integrated into the *ccf* process of the **IN_ssp** model. The throttling algorithm will be described in Section 4.2.2.1, including justifications for why it does not reflect exactly the description of the throttle provided by Bellcore [Bellcore92]. The PT throttle (Section 4.2.2.2) is also located in the *ccf*. Both throttles have 15 throttling levels defined.

4.2.2.1 The CG Throttle Mechanism

The CG throttle implemented in the model does not correspond to the throttle that was standardised by Bellcore. The primary reason for this, as mentioned in Chapter 2, is that investigations early in the development of the model proved that the gap durations and gap interval levels defined by Bellcore were impractical to use. The times specified were far too long and gave bad results. This is corroborated by [Smith95]. However, some of the ideas presented by Bellcore were very useful and were used as guidelines when implementing the throttle.

The CG throttle algorithm is included in the *ccf* process of the SSP model. Fifteen levels of CG control are defined here, and their relationship to the overload levels defined by the detection algorithm in the SCP is maintained using a table of gap interval levels corresponding to the overload levels. Suitable gap interval times were achieved through trial and error to maximise throughput during congestion, while ensuring recovery was as rapid as possible. Note that finding appropriate gap interval times is a non-trivial task.

Each time the *ccf* of the SSP receives an overload message from the SCP signifying a change in overload level, the gap interval level under which the throttle was operating is changed accordingly, by resetting all gap timers, evaluating the gap interval level associated with the new overload level and altering the gap timers to these new levels. When the next request arrives after the controls have been reset, the gap timer is set for the length of the new gap interval and all calls arriving while the timer is active are rejected. When the timer expires, the next arriving call is accepted and the timer is reset. Throttling continues at this level until the next control message arrives from the SCP, detailing a change in the overload level.

Note that as all changes in overload level are relayed to the SSP immediately after detection in the SCP, gap interval levels were designed to remain in place until altered by the arrival of a new overload message and therefore duration levels were not required in the model and were omitted. Also, as all detection methods caused the same overload message to be transmitted to both SSPs, both were throttled equally.

4.2.2.2 The Percent Thinning Throttle Mechanism

The PT throttle algorithm is also included in the *ccf* process of the SSP model, but it is not necessary to maintain a table of PT coefficients corresponding to the SCP overload levels here, as the appropriate PT coefficient is sent by the SCP to each SSP as part of the overload notification (the SCP divides the measured value (e.g. for CCC, the number of new arrivals) by the measured value associated with the SCP threshold in order to evaluate the PT coefficient). The PT throttle

then applies the new PT coefficient to arriving traffic according to the algorithm shown in Figure 4.13, in order to allow the acceptance of only the relevant percentage of arrivals.

```
Increment new_calls;
If current_accepts/new_calls <= PT_coefficient {
     Accept call;  increment current_accepts; }
Else reject call.
```

Fig. 4.13: The PT algorithm

Each time the *ccf* of the SSP receives an overload message from the SCP signifying a change in overload level, the counters in the PT algorithm (i.e. `current_accepts` and `new_calls`) are initialised and the new percent thinning coefficient is applied. Thereafter, each time a new call arrives, `new_calls` is incremented and the algorithm is used to evaluate whether the call should be accepted or rejected. The algorithm continues using any given PT coefficient until the next overload notification arrives from the SCP, at which point the algorithm is re-initialised with the new coefficient.

Note that while the SCP sends the same throttling coefficient to both SSPs, the use of the PT algorithm ensures that arrivals at each SSP will be throttled proportionally to their arrival rates, and therefore, unlike CG (which will place the same strict upper limit on the number of calls to be accepted at all SSPs), acceptance rates at each SSP will be proportional to the number of requests arriving at that SSP.

4.2.3 Implementation of the Window Strategy

According to the Window mechanism described in Chapter 2, throttling of calls should take place both at the SCP and the SSP. However, rejection of calls at the SCP is undesirable for two reasons – firstly, SCP throughput should be maximised during overload and therefore SCP processor resources should not be expended rejecting calls and secondly, each SSP would have to make assumptions about whether or not a call was rejected by the SCP (as no notification of overload is sent to the SSPs) and making a wrong guess at any point would lead to livelock within the SSP Window algorithm. Therefore, an adapted version of Window is applied, in which no calls are rejected at the SCP, and the SSPs alone are responsible for ensuring that no SCP overload takes place. Therefore the Window algorithm of each SSP, located at the *ssf* process of each SSP, monitors the response time of the SCP to requests from that SSP. When response delays become excessive, calls are throttled.

A Window size counter W is defined and initially takes on a pre-defined value Wmin. W corresponds to the maximum number of new call queries for which an initial response is

outstanding from the SCP (note that some services may consist of multiple SSP-SCP query/response pairs). Each time a new query is sent to the SCP, a variable *OUT* (signifying the number of calls for which a response is outstanding) is incremented and a timer is set (the value of the timer duration was established from observation of the delays which occurred during simulation when the SCP load was at the threshold). If $OUT=W$, all new queries are rejected at the SSP until a response is received from the SCP. Each time a response is received from the SCP, a variable C is incremented (indicating a positive response), *OUT* is decremented and the corresponding timer is reset. If a timer expires, this signifies that the delay experienced by the corresponding call is greater than the maximum acceptable level, thus implying that congestion exists (either within the SS7 or at the SCP). When this occurs, the window size W is decremented to force a greater level of throttling of IN calls and C is reset to zero. When C has increased to the point that it exceeds a pre-defined value Cmax, this is interpreted as alleviation of the overload situation, and the window size W is incremented. In this way, the SSP responds to SCP overload without explicit communication between the physical elements.

4.3 Presentation of Results

In this section, we present the results for each of the given strategies under an applied load consisting purely of freephone traffic – the effects of multiple traffic types on the strategies is a separate issue, which will be examined in Section 4.4. In the first part of this section, we prove the need for congestion controls by comparing the behaviour of the system under overload when no controls are in place with the behaviour when CCC and CG are used. Then, in Section 4.3.2, the various detection methods of the reactive (communication-oriented) strategies are compared to establish which is the most efficient at protecting the SCP. Note that, to ensure fairness of comparison, CG was used as the throttling mechanism in all cases. In Section 4.3.3, the CG and PT throttles are compared to find out which one is most accurate at rejecting the desired proportion of arrival calls during overload. This investigation was carried out using CCC in both cases – namely, the detection method which provided the best performance in Section 4.3.2. At this point, it will be possible to state which reactive strategy is the best, in terms of protecting the SCP most effectively. Finally, the active (communication-less) Window strategy is compared with the best reactive strategy in Section 4.3.4 to find which of the existing strategies has the best overall performance.

4.3.1 Proving the Need for Congestion Controls

Here, we detail the findings of a simulation where the input (freephone) traffic was increased linearly over the course of a simulation, in a manner sufficient to cause overload at the SCP after

about 470 seconds. When no congestion controls are in place, the SCP quickly becomes saturated and a serious backlog of calls builds up at the SCP, with the result that the SCP queue length grows exorbitantly (Figure 4.14) and the user delays quickly become untenable (Figure 4.15). This proves the need for congestion control strategies to protect the SCP.

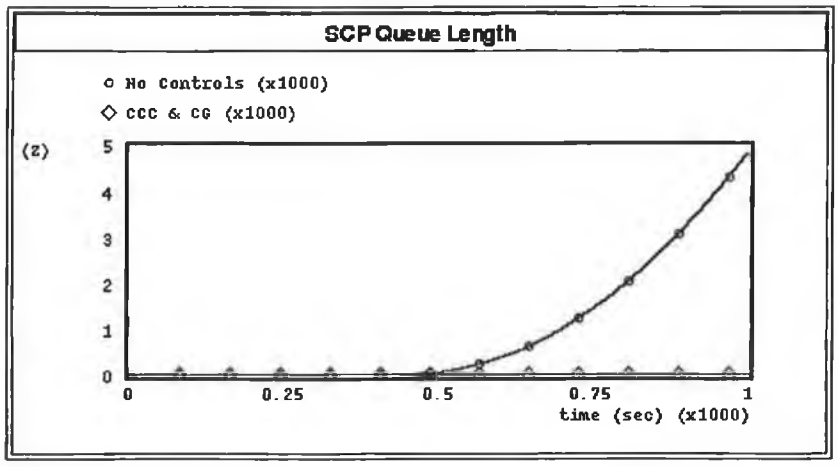


Fig. 4.14: SCP Queue Length for CCC/CG vs No controls in SCP

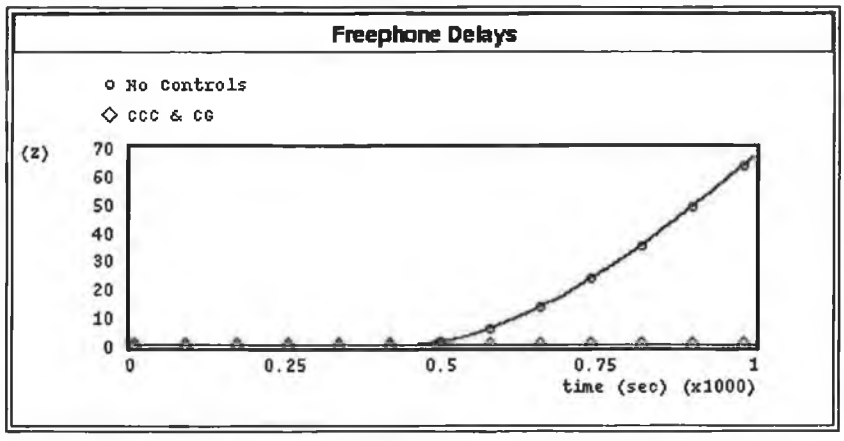


Fig. 4.15: Service Delays for CCC/CG vs No controls in SCP

4.3.2 Comparison of Detection Methods for Reactive Communication-Oriented Control

Here we compare the four implemented detection methods to establish which one provides the best consistent behaviour across all load situations. We therefore apply three different input freephone traffic scenarios – namely constant mean, linearly increasing mean and bursty traffic. In all cases, the same CG throttle, SCP and SSP service rates are used, to ensure that the comparison is strictly between detection methods. Also for all cases, the SCP load threshold is defined as being 0.8 Erlangs and the monitoring interval as 10 seconds.

4.3.2.1 Stationary Behaviour

As a first step, we investigated the behaviour of the various detection methods in the stationary case, i.e. when input traffic levels have a constant mean for the duration of a simulation and are at a level sufficient to cause overload at the SCP - Figure 4.16 shows that the input traffic is sufficient to offer the SCP a mean of 1.4 Erlangs of work.

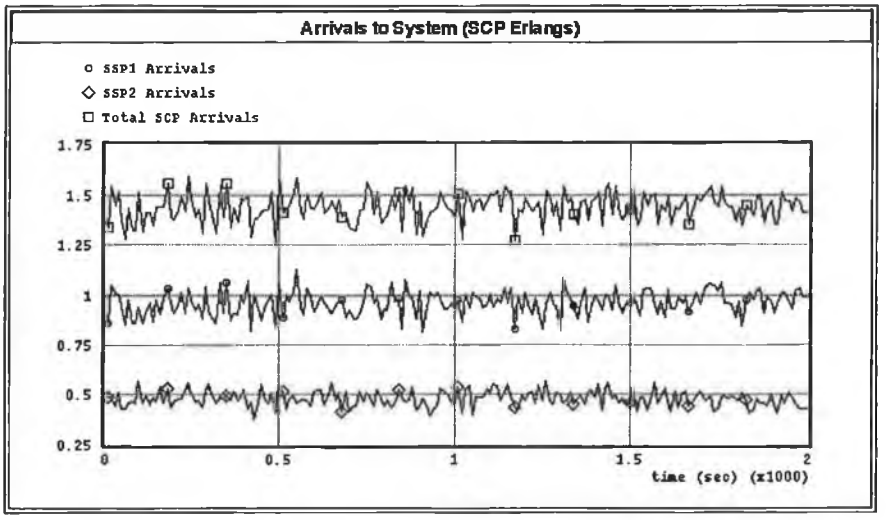


Fig. 4.16: Arrivals to system for stationary case

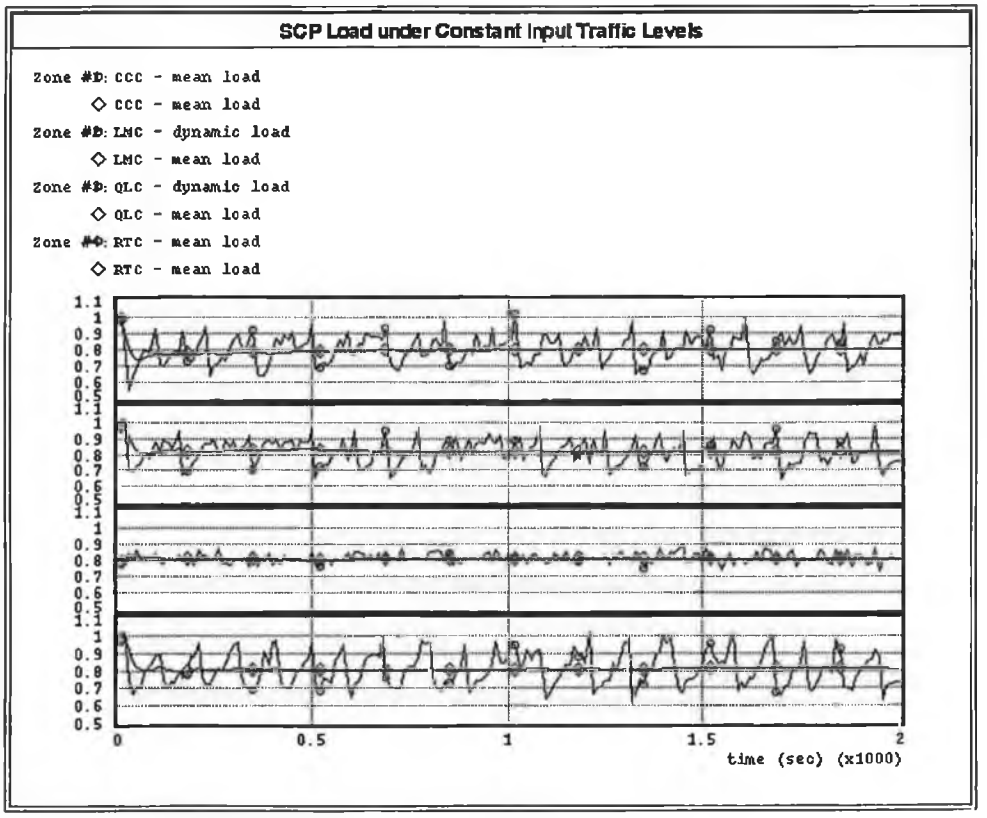


Fig. 4.17: SCP load for stationary case

Figure 4.17 presents the resultant SCP load over the course of the simulation. A number of facts are noteworthy about these results. Firstly, all detection methods (in conjunction with the same CG controls) succeed in protecting the SCP, by keeping the load below 1.0 Erlang and secondly, all detection methods fail to converge to a particular overload level, but instead experience oscillations of differing size around a mean of 0.8. The oscillations may, in part, be explained by the oscillations around the mean of the input traffic, but are also affected by the detection methods used. Note that for QLC, which is not based on monitoring the system over an interval but instead reacts immediately when overload is detected, the oscillations are small and the mean SCP load converges quickly. Of the methods based on the use of monitoring periods, CCC and LMC have smaller oscillations in the dynamic load and converge more quickly in the mean load than does RTC – this is because both CCC and LMC overload levels have a linear relationship with the SCP arrival rates whereas RTC, whose overload levels are based on delays, and therefore by extension on queue lengths, has a non-linear relationship with the SCP arrival rates. However, all strategies converge to a mean of 0.8, the pre-defined SCP load threshold.

Figure 4.18 shows the mean queue length at the SCP over the course of the simulation. As would be expected, QLC responds immediately to any rise in queue length, and therefore no oscillations occur in SCP mean queue length for this method. For the other methods, the delay before overload is detected (i.e. the monitoring period) means that an excessive number of calls are accepted originally and must be processed before the SCP load and queue length stabilise. Therefore, at the start of the simulation, the queue length rises to over 100, before gradually dropping to acceptable levels (this is more a measure of how quickly these methods respond to dramatic increases in traffic than a measure of their steady state behaviour). Note that CCC, LMC and RTC have very similar behaviour here, with CCC being only very slightly faster at reaching a stable queue length than the other two strategies.

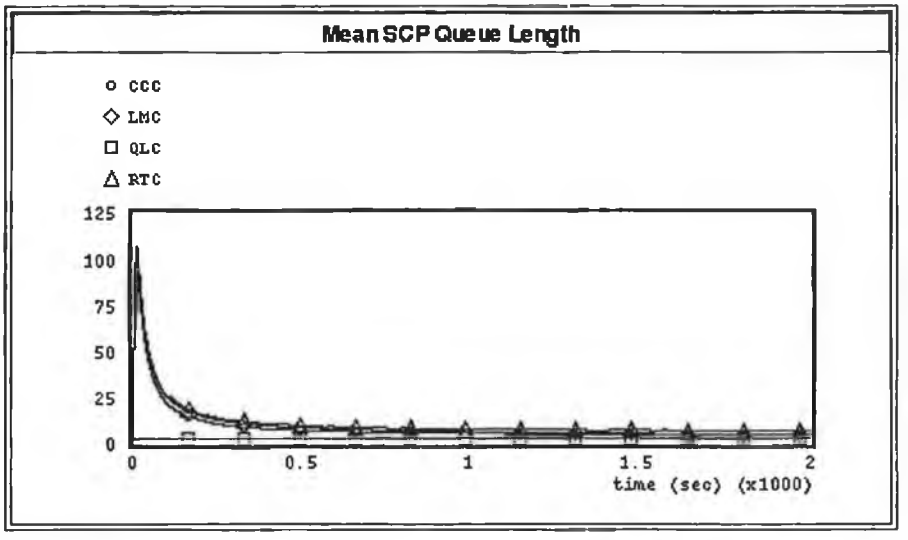


Fig. 4.18: SCP queue length for stationary case

The conclusion of this is that QLC has the best performance in steady state, in that it experiences smaller oscillations than the other methods, although to achieve this, its overhead in terms of SCP processor utilisation is a factor of 2.1 greater than the other strategies. This is due to the fact that it remains active at all times. The other methods, on the other hand, provide satisfactory results while only being active at the end of each monitor interval.

4.3.2.2 Behaviour under Linear Increase in Arrival Rates

To investigate the behaviour of each of the detection methods under rapid increases in input traffic, as well as to establish their behaviour at low and high overload levels, the mean arrival rate of freephone traffic was increased linearly, as shown in Figure 4.19. Note that arrivals to the system increase from zero to 4.6 SCP Erlangs over the course of a simulation and that a load of 1.0 Erlangs is provided after about 400 simulated seconds.

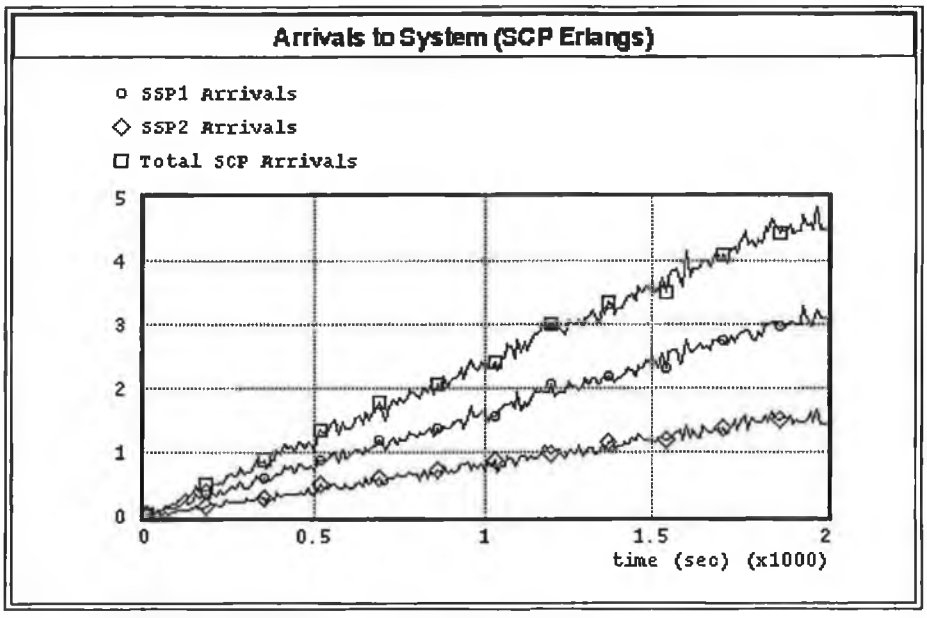


Fig. 4.19: Arrivals to system for linearly increasing freephone arrival rate

The resultant variations in mean SCP load for each of the methods are shown in Figure 4.20, while the dynamic variations in SCP load are shown in Figure 4.21.

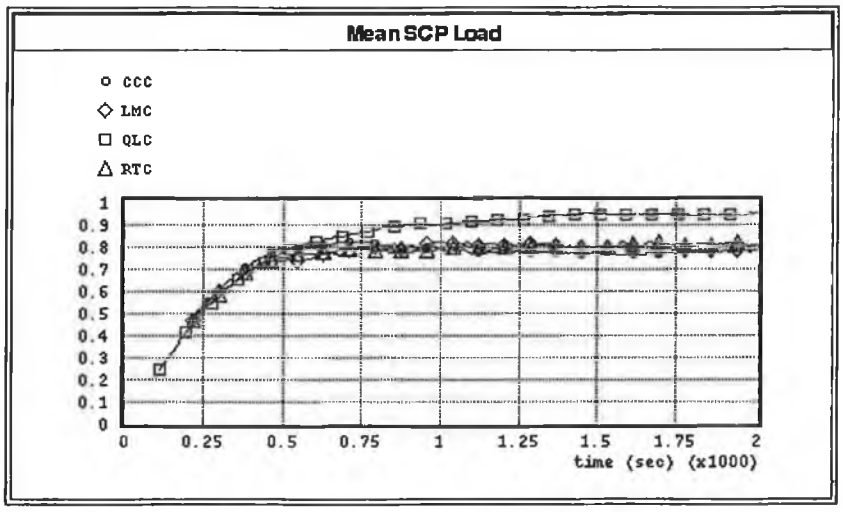


Fig. 4.20: Mean SCP load for linearly increasing freephone arrival rate

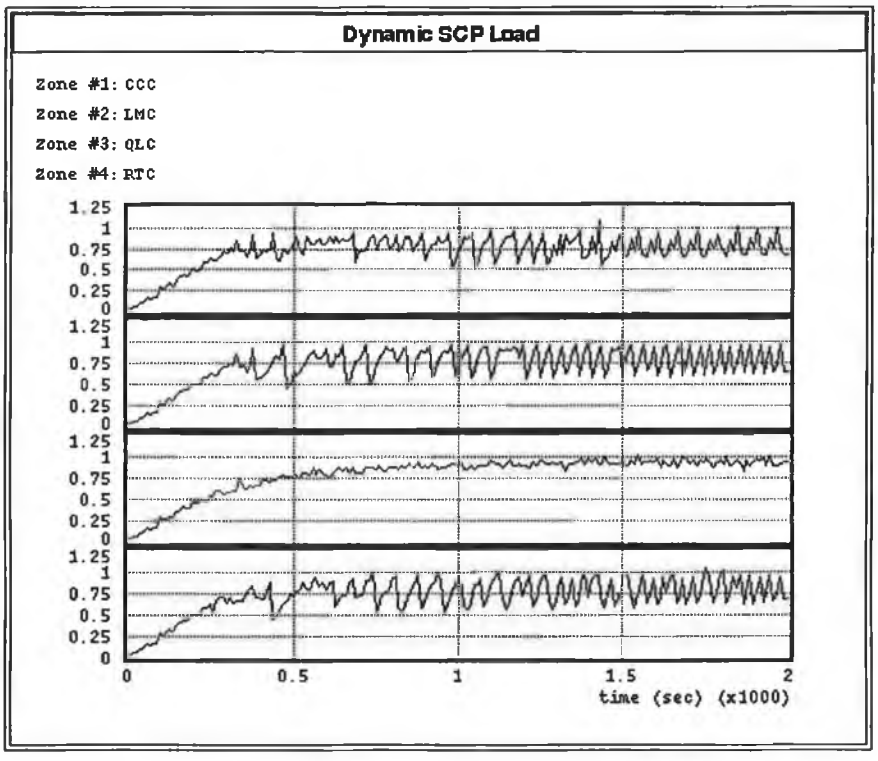


Fig. 4.21: SCP Load for linearly increasing freephone arrival rate

The first comment to be made is that none of the strategies provide satisfactory performance over all load levels – CCC, LMC and RTC load levels experience large oscillations around the threshold, while QLC permits overload to take place at high applied load levels. The reason for the oscillations experienced by all detection methods is related to the fact that these methods are based on a table of fixed overload parameters. When a detection method is based on fixed parameters, it tends to be prone to oscillations (this was even apparent in the stationary case of section 4.3.2.1) as it does not respond to exactly the applied load level, but instead rounds to the overload level corresponding to the input data (call count, service delays etc.). This means that each detection method tends to swing from overprotecting to underprotecting the SCP on alternate monitoring

intervals. The size of the oscillations is defined by the parameter values, but even when these values are very carefully defined and quite precise, oscillations will still occur. This means that it always takes time for a detection method to converge to the correct overload level (or even to reach minimal oscillations between load levels). When the traffic is consistently increasing, as in this scenario, none of the detection methods have time to converge – they oscillate dramatically and for each interval where the SCP is underprotected, the SCP queue length increases slightly, as demonstrated in Figure 4.22.

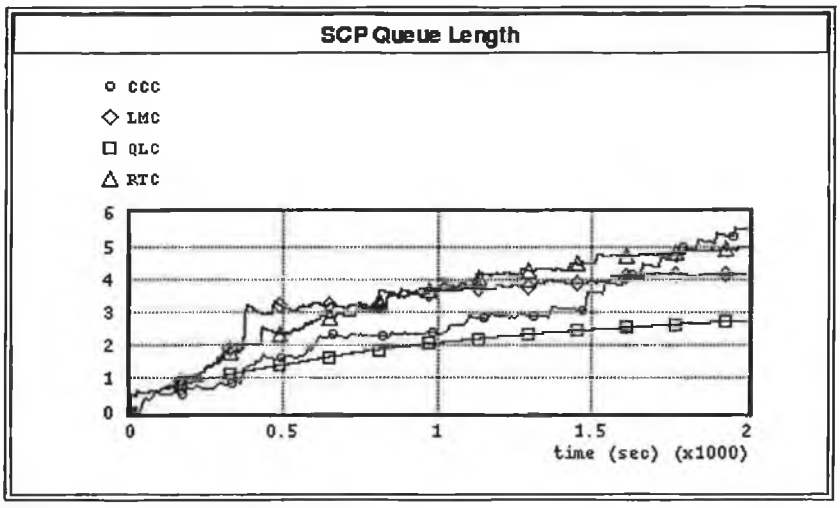


Fig. 4.22: Mean SCP queue length for linearly increasing freephone arrival rate

While it is accepted that all strategies are innately flawed due to their dependence on fixed parameters, it is still possible to compare their behaviour to establish which provides the best performance (and how). Regarding onset of congestion, examination of the dynamic load results in Figure 4.21 shows that all methods detect overload too early. CCC and LMC show the best results, by detecting overload when the actual submitted load is 0.78 Erlangs. For CCC, this is probably due to minor variations in the arrival rate during that monitoring period. For LMC, note that its response to overload does not occur as early as is described for LMC when applied to a switching system (at 60% capacity, as described in Section 2.2.3.2) – this is because in a switching system, processing of initial requests takes only 30% of the overall processing time required for a call, whereas for freephone, initial processing at the SCP takes 50% of overall SCP processing requirements and therefore, for freephone, the load estimate reached by LMC is more representative of arriving requests and therefore more accurate. Note however, that for other call types (e.g. televoting, which requires SCP processing four times per call), LMC would be less accurate. QLC, due to its overly reactive nature, starts responding to overload when the submitted load is only at 0.62 Erlangs – i.e. bursts of traffic at this level are sufficient to be construed as overload by QLC. RTC responds when the input traffic is at about 0.7. This is because service delays are very dependent on SCP queue lengths and therefore, occasional short increases in the queue length result in increased average service delays which then trigger overload controls early.

The behaviour of each of the detection methods under overload may be summarised as follows:

- CCC is located at the input of the SCP and is based on counting the number of newly arriving calls during an interval. It is therefore capable of responding very quickly to the onset of overload. Also, the number of new arrivals is a very accurate representation of the applied load, with the result that CCC experiences slightly smaller oscillations than LMC or RTC.
- LMC's reaction to the onset of overload is delayed as its calculation of overload level is based not only on the amount of time it has spent processing new requests but also on the load required by old requests (returning from the SDP). It therefore takes longer for LMC to accurately detect overload, as the full effects of the overload are not felt until old requests return from the SDP for further processing. This delay in overload detection means that it also takes LMC slightly longer to recover from overload, as it must complete the successive processing requirements of all old requests which received initial processing between the time overload occurred and the time overload is detected (note that, as a result of this, the SCP queue length tends to be longer for LMC than for CCC).
- RTC is based on measuring the mean service execution time for requests of each IN service type. This means that there is a significant delay between the onset of congestion and its detection by RTC, as an overload is allowed to propagate through the system until the queues have grown sufficiently long at the SCP, SDP and IP to significantly affect the mean response times for service requests. As a result of this, RTC tends to have the largest mean SCP queue length and by extension, the longest average service delays (shown in Figure 4.23). RTC's response delay (as with LMC) has further implications, in that all requests that were accepted during this delay must complete processing and recovery time is slower than for CCC.

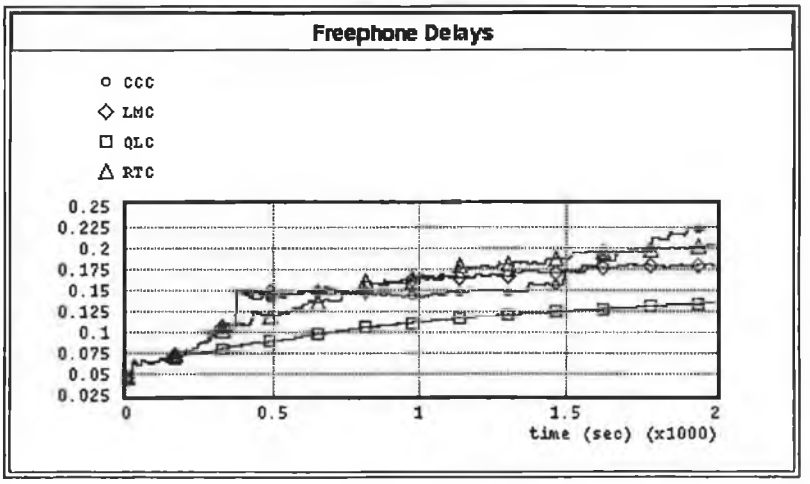


Fig. 4.23: Mean freephone delays for linearly increasing freephone arrival rate

- At low overload levels, QLC succeeds in protecting the SCP from overload. However, it does not converge, but instead allows the load to increase slightly and non-linearly as the applied traffic increases. Note also that QLC exhibits minimal oscillations – this is because it is not

based on the use of a monitoring interval. Instead, it responds immediately to increases in queue length and therefore has a tighter feedback loop than the other strategies. However, as was mentioned above, this means that QLC has a tendency to be too reactive – it responds to minor fluctuations in applied load by putting unnecessary controls in place. A further negative implication may be associated with this tight control loop – while CCC, LMC and RTC reset the CG throttle with new coefficients at most once per monitor period, QLC attempts to reset the throttle every time the queue length changes which, when the SCP is nearing saturation (i.e. arrival rate to the SCP is much greater than the service rate), is nearly every time a new requests arrives. This effectively renders the throttle impotent and so the number of calls arriving at the SCP rises dramatically. The eventual result is that the SCP becomes saturated.

The conclusion of this is that CCC is the best strategy for linearly increasing traffic – it does not respond too early to the onset of congestion, and protects the SCP under all traffic loads with smaller oscillations and shorter queue lengths than either LMC or RTC. LMC's operation is nearly as good, but has greater response delays (and therefore longer queue lengths) than CCC. RTC, while capable of protecting the SCP, exhibits greater oscillations and longer queue lengths and delays than either LMC or CCC. QLC shows extremely undesirable behaviour – it both responds too early to overload and can only protect the SCP at low overload levels.

4.3.2.3 Behaviour under Bursty Traffic Input

To investigate the behaviour of each of the methods under bursty traffic input, the arrival rates to the system were defined as shown in Figure 4.24 – note that these are expressed in terms of SCP capacity.

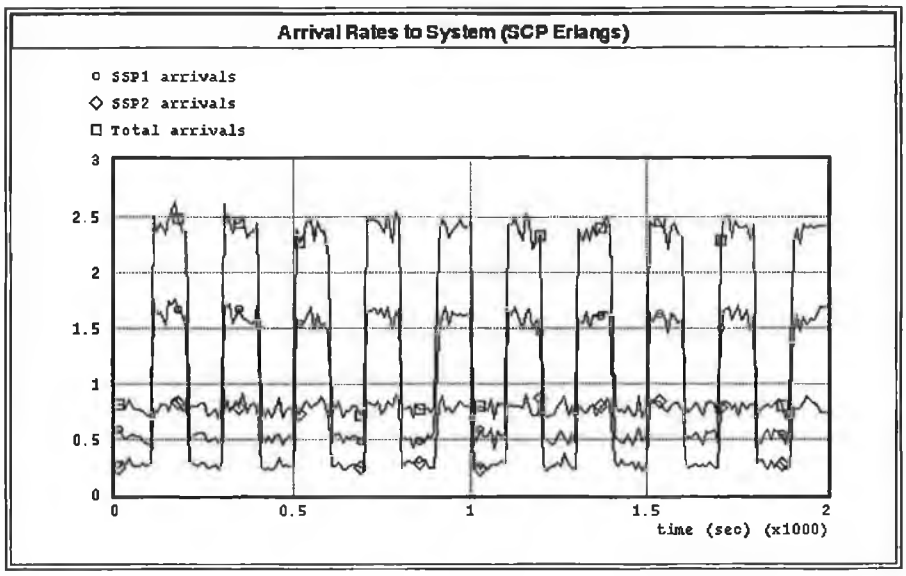


Fig. 4.24: Arrivals to system for bursty arrival rates

The resultant dynamic SCP loads for each method are shown in Figure 4.25.

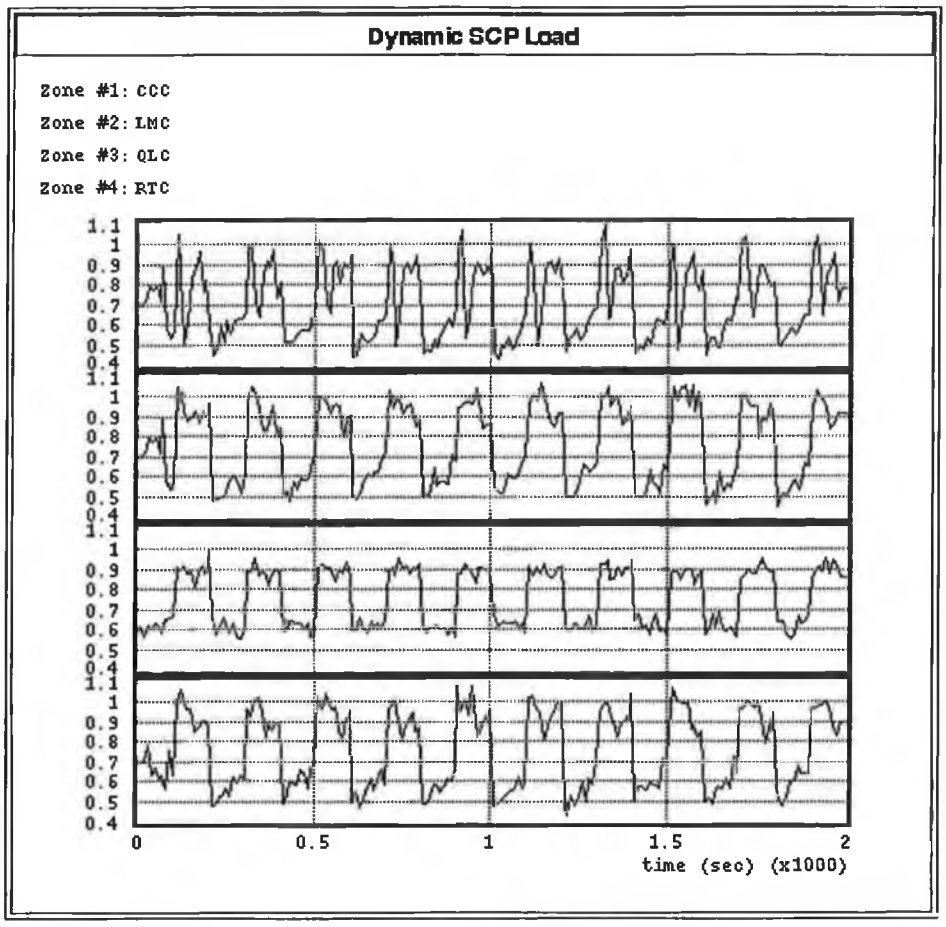


Fig. 4.25: SCP load for bursty arrival rates

Note that, as expected, QLC is the only method that is reactive enough to protect the SCP at all times from burst input traffic – for this method, the SCP load rarely approaches 1.0 Erlang. For the other strategies however, overload is not detected until the end of a monitoring interval and therefore, the SCP load rises to greater than 1.0 Erlang for each large input traffic burst. CCC then responds very quickly by bringing the SCP load down to a more acceptable level. LMC and RTC on the other hand, react too slowly and the SCP load remains at unacceptably high levels for a few monitoring intervals, before being reduced. Note also that QLC provides a greater mean SCP load over the course of the bursty simulation, giving a mean of 0.75 Erlangs, as opposed to CCC's mean of 0.705 Erlangs.

These results are also reflected in the SCP queue length (Figure 4.26) and freephone delays (Figure 4.27). Note that only QLC and CCC provide freephone delays which are within acceptable bounds, as defined by [E.723] and [MacDonald94].

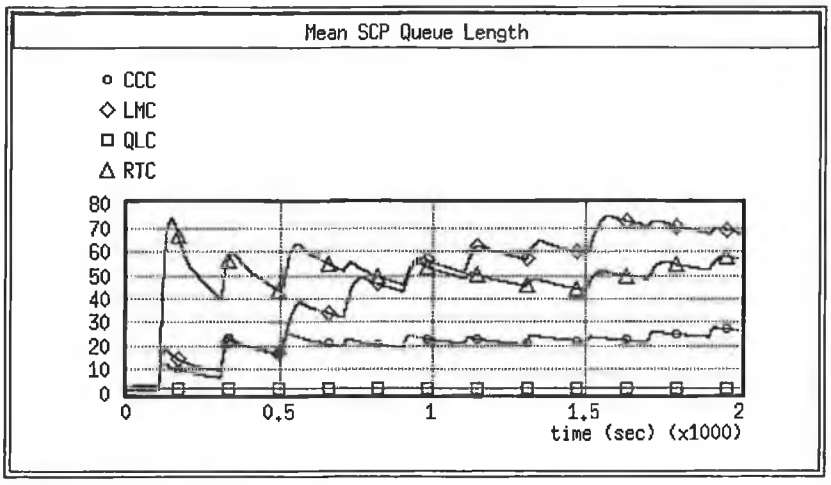


Fig. 4.26: Mean SCP queue length for bursty traffic

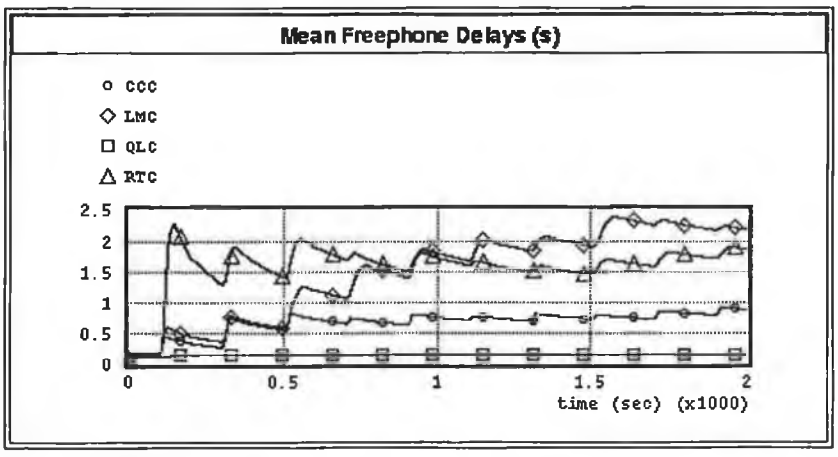


Fig. 4.27: Mean freephone delays for bursty traffic

4.3.2.4 Summary of Detection Method Results

The salient features of each of the detection methods is outlined in Table 4.1 below, where a √ denotes acceptable behaviour and a * denotes best behaviour for each category.

Category	*CCC*	*LMC*	*QLC*	*RTC*
Steady state behaviour	√	√	√ (*)	√
Relative processor requirements	1	1	2.1	1
Correct response to onset of congestion	√	√		
Correct response to low overload	√ (*)	√	√	√
Correct response to high overload	√ (*)	√		√
Correct response to bursty traffic	√		√ (*)	

Table 2.1: Summary of detection method results

The obvious conclusion to be drawn from this summary is that none of the strategies provide acceptable results over all possible input scenarios, but CCC generally provides the best results. The only method which ever performs better is QLC, and even in those cases, CCC provides acceptable results and outperforms the other two methods. As a result, we select CCC as being the detection method that provides consistently the best behaviour and will therefore use it in conjunction with both the CG and PT throttles to establish the best possible reactive strategy.

4.3.3 Comparison between Throttles

Here we compare the operation of the Call Gapping and Percent Thinning throttles to establish which one provides consistently the best behaviour across all load situations, using the same input freephone traffic scenarios as in Section 4.3.2 – namely constant mean, linearly increasing mean and bursty traffic. In all cases, the same CCC detection method, SCP and SSP service rates are used, to ensure that the comparison is strictly between throttles. Also for all cases, the SCP load threshold is defined as being 0.8 Erlangs.

4.3.3.1 Stationary Behaviour

As a first step, we investigated the behaviour of the throttles in the stationary case, i.e. when input traffic levels have a constant mean for the duration of a simulation and are at a level sufficient to cause overload at the SCP - the input traffic offered is as shown in Figure 4.16. The resultant dynamic SCP load is as shown in Figure 4.28, while the mean SCP load is shown in Figure 4.29.

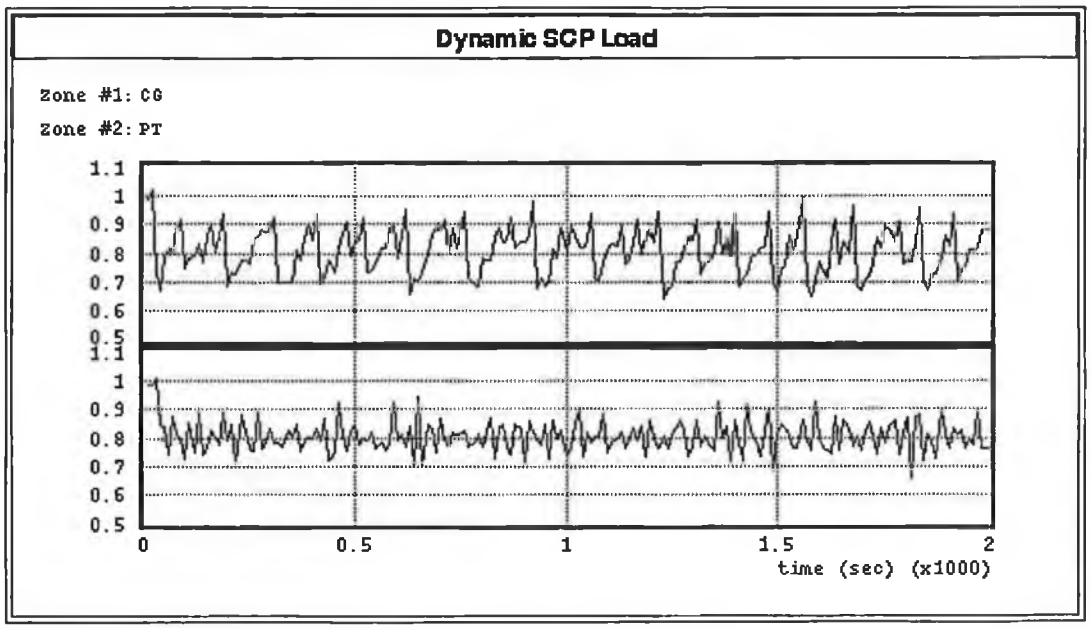


Fig. 4.28: Dynamic SCP load for stationary case

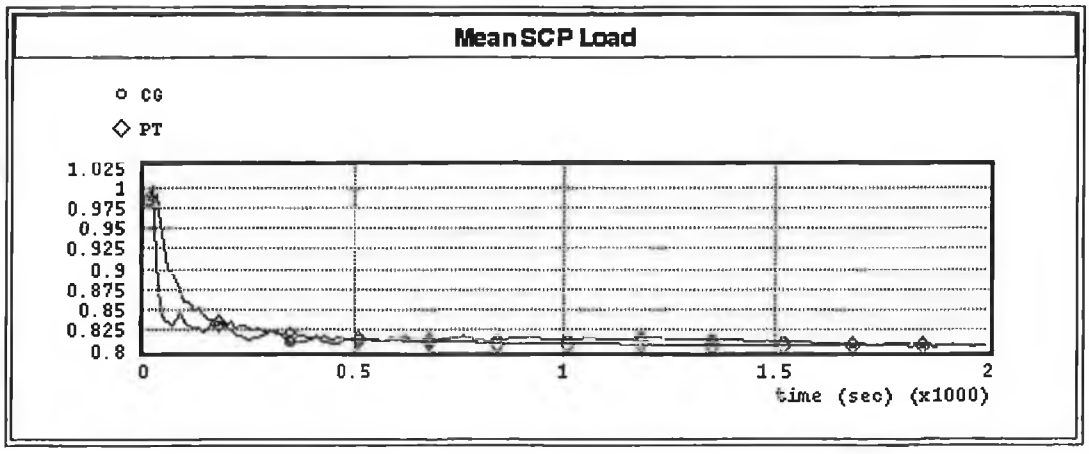


Fig. 4.29: Mean SCP load for stationary case

Note that the oscillations in the dynamic SCP load are much smaller for PT than for CG – this is due to its dynamic nature – i.e. the throttle put in place reflects accurately the current overload condition. For table-driven CG, however, the throttle reflects the table entry closest to the current overload condition and not the overload condition itself, resulting in greater oscillations over the course of the simulation. On the other hand, the mean SCP load shows that CG reacts more quickly to an overload, bringing the SCP load down to the threshold much faster than PT. This is because when a CG throttle is put in place, it places a strict upper limit on the number of calls which may be accepted in the following monitoring interval and therefore makes the system more robust to increases in call arrivals during that period. PT fails to do this, as it merely accepts a fixed percentage of the arrivals in the following period. The faster reaction time of CG is also reflected in the mean SCP queue length, as shown in Figure 4.30.

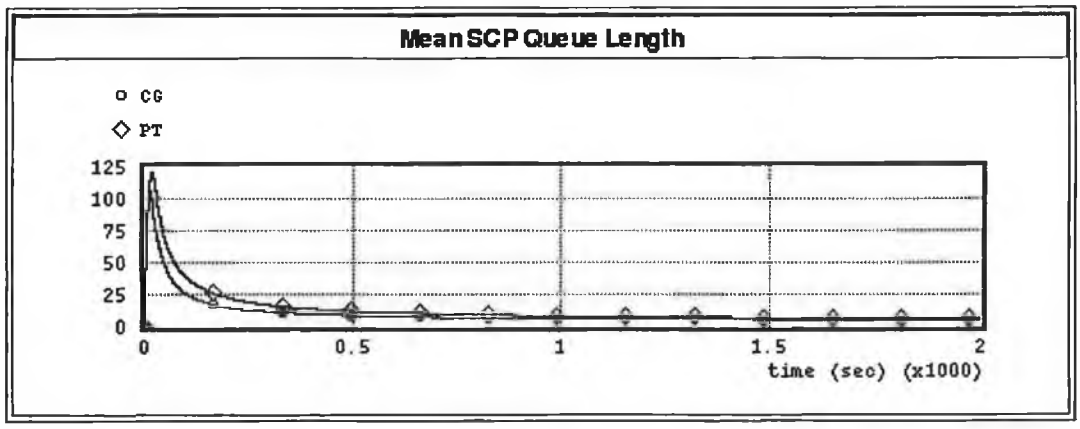


Fig. 4.30: Mean SCP queue length for stationary case

The greatest difference between the behaviour of CG and PT, however, may be observed by viewing the ratio of call acceptances for each of the SSPs. Figure 4.31 shows the call acceptances for SSP1 and SSP2 for the stationary case. Note that for CG, SSP1 (which receives twice as many IN calls as SSP2) has a much lower acceptance rate than SSP2. This is because the same gap interval is put in place in both SSPs and, as SSP1 has a greater arrival rate, more calls are rejected

– in fact, the overall result is that the same number of calls are accepted by both SSPs. For PT, on the other hand, the same thinning coefficients are put in place at both SSPs, resulting in the same acceptance rates at each, and therefore the ratio of calls which arrive at the SCP from the SSPs is maintained (i.e. SSP1 both receives and accepts twice as many IN calls as SSP2).

The conclusion of this is that, in the stationary case, CG is more robust and faster at responding to overload than PT, while PT retains the ratio between arrival rates from each SSP to the SCP (i.e. it exhibits subscriber fairness).

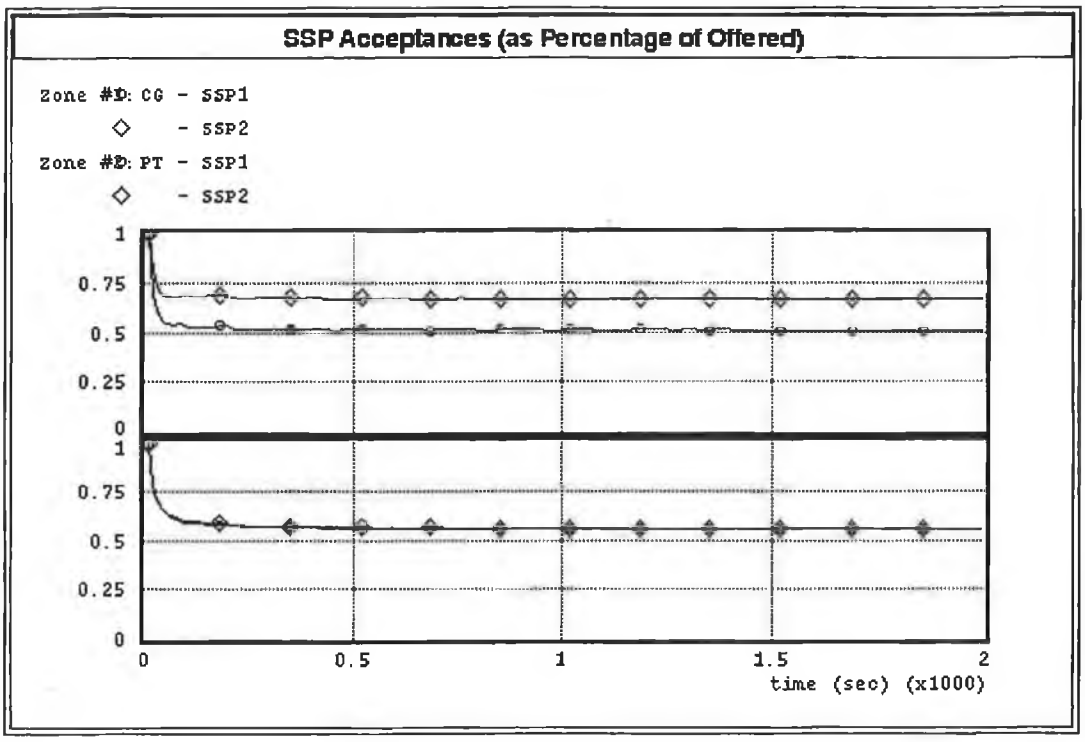


Fig. 4.31: SSP acceptances for stationary case

4.3.3.2 Behaviour under Linear Increase in Arrival Rates

To investigate the behaviour of each of the throttles under rapid increases in input traffic, as well as to establish their behaviour at low and high overload levels, the mean arrival rate of freephone traffic was increased linearly, as shown in Figure 4.19. The resultant dynamic SCP load is similar to that shown in the stationary case – i.e. CG exhibits much greater oscillations over the course of the simulation than PT. Again, this is due to the fact that the throttles put in place by PT more accurately reflect the state of the SCP than those put in place by CG. However, unlike the stationary case, this does have an impact on the mean SCP load, as shown in Figure 4.32. Note that CG has a tendency to overprotect the SCP. This is a direct result of its table-driven nature – the immediate reaction of CG to overload is to overprotect the SCP and while given enough time (as in the stationary case), this will eventually converge to the SCP threshold, if variations in arrival rates occur over a number of monitoring intervals, CG will fail to converge and the SCP will

remain overprotected. This is undesirable, as calls are being rejected unnecessarily. PT, on the other hand, tends to underprotect initially (as described in the stationary case), but compensates fast due to its dynamic nature and therefore maintains the SCP load either at or slightly above the threshold.

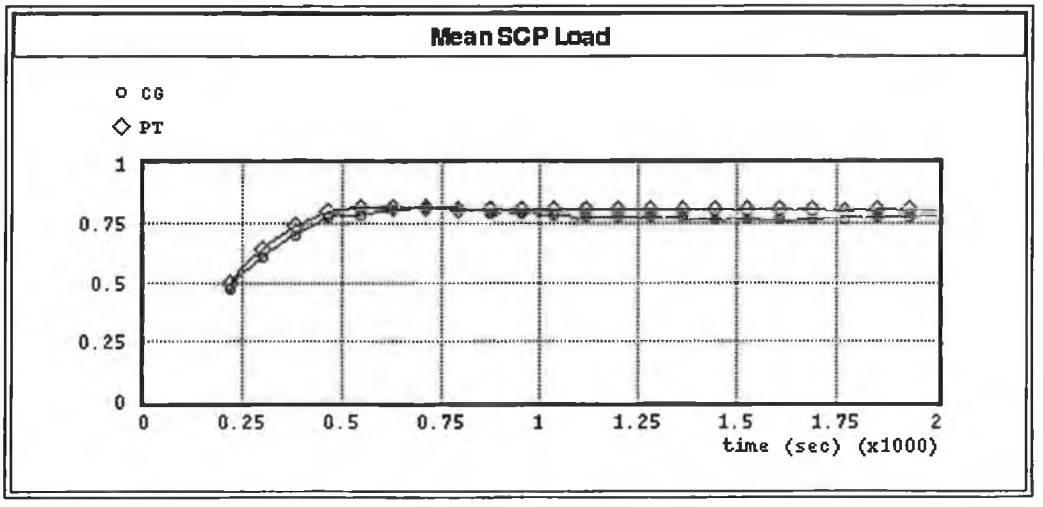


Fig. 4.32: Mean SCP load for linearly increasing arrival rates

To conclude, both CG and PT protect the SCP at all times from overload, although PT's behaviour is more consistent, due to its dynamic nature. Also, again, only PT exhibits subscriber fairness, as shown in Figure 4.33.

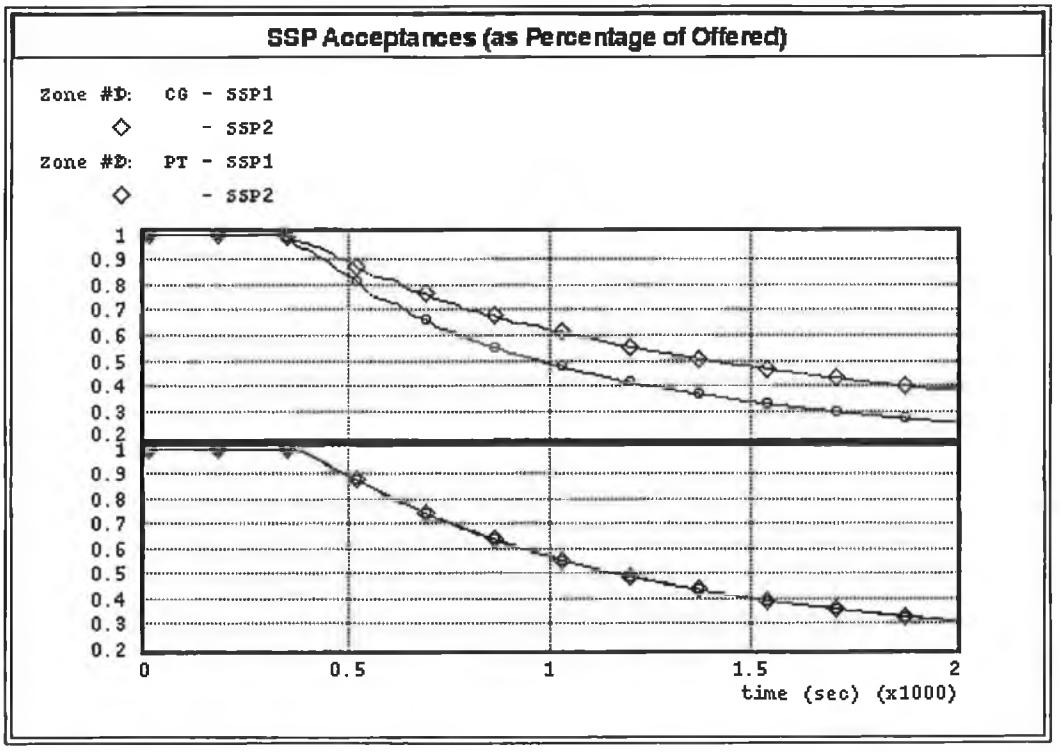


Fig. 4.33: SSP acceptances for linearly increasing arrival rates

4.3.3.3 Behaviour under Bursty Traffic Input

To investigate the behaviour of each of the methods under bursty traffic input, the arrival rates to the system were defined as shown in Figure 4.24. The resultant dynamic SCP load is shown in Figure 4.34.

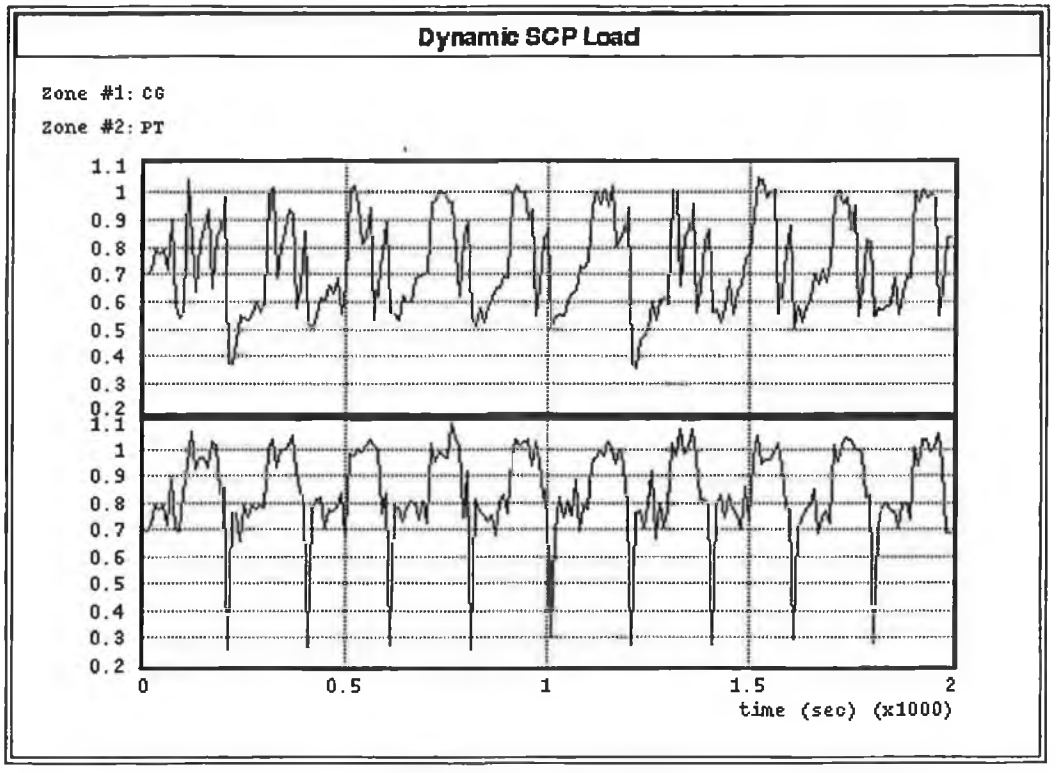


Fig. 4.34: Dynamic SCP load for bursty arrival rates

This graph shows that an instantaneous increase in arrival traffic causes SCP load to jump to over 1 Erlang and the SCP queue length to increase dramatically – this is as a result of the monitoring delay associated with CCC. When CG is invoked, it responds rapidly by putting excessive throttles in place (excessive because it is based on fixed parameters), thus generally giving the SCP time to process the calls in the queue and alleviate the overload condition during the next interval. PT, on the other hand, puts exactly the correct proportional throttles in place on detection of overload and therefore does not give the SCP time to process the call requests which had built up in the queue during the previous interval. It therefore fails to alleviate overload quickly, the SCP load remains at approximately 1 Erlang for the entire duration of the burst and the mean SCP queue lengths remain substantially higher for PT than for CG for the entire duration of the simulation – see Figure 4. 35. So, in this case, PT actually suffers due to its accuracy.

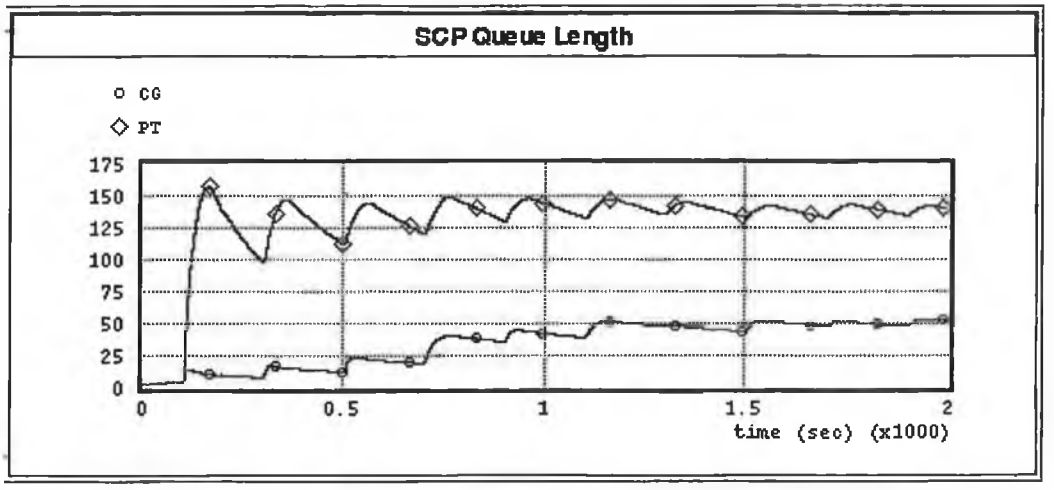


Fig. 4.35: Mean SCP queue length for bursty arrival rates

When a bursty period ends, both strategies overprotect the SCP until the end of the next monitoring interval, at which point both strategies recover quickly, with PT actually converging faster to the mean load of 0.78.

The resultant service delays for this scenario are shown in Figure 4.36. Note that the delays for CG are near acceptable limits, as defined by [Yan94], while the delays experienced by calls under the PT throttle are clearly unacceptable.

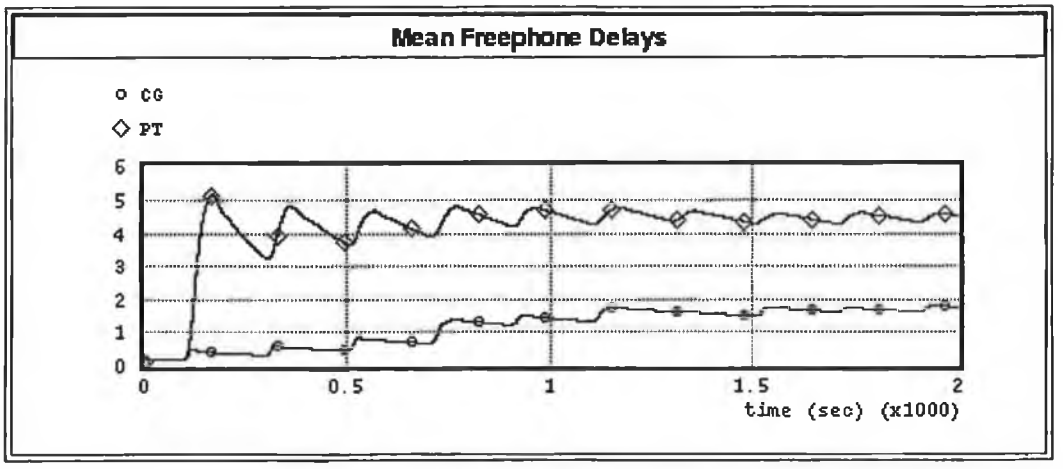


Fig. 4.36: Mean freephone delays for bursty arrival rates

The conclusion of this is that CG provides a better instantaneous response to dramatic increases in input traffic, as its tendency is to overprotect the SCP while PT's tendency towards accuracy means that the SCP is vulnerable to rapid increases in traffic when PT is used. PT, however, is faster to converge to an optimal level than CG. This is not necessarily useful after a rapid increase in traffic, as PT's slowness in responding generally causes a large build-up of the SCP queue, which then takes further time to serve. However, PT's speed of convergence does maximise SCP throughput after a rapid decrease in traffic.

4.3.3.4 Summary of Throttle Results

The salient features of each of the throttles is outlined in Table 4.2 below, where a √ denotes best behaviour for each category.

Category	*Call Gapping*	*Percent Thinning*
Relative processor requirements	4.4	1
Rapid response to onset of congestion	√	
Rapid response to end of congestion		√
Accuracy (speed of convergence)		√
Subscriber fairness		√

Table 2.2: Summary of throttle results

To summarise, the principle advantage of the CG throttle is that it places a strict upper limit on traffic acceptances rates, which in the short term means that it responds better to rapid onset of congestion. However, the dynamic PT throttle converges faster to the threshold and therefore provides better results in the long term. It has the added advantage of being subscriber fair, in that it throttles all sources proportionally to their size. It may therefore be concluded that the ideal throttle would combine CG's speed of response with PT's accuracy and fairness.

4.3.4 Active versus Reactive Congestion Controls

In this section, the behaviour of the active communication-less Window congestion control algorithm is compared with that of the reactive communication-oriented CCC/CG and CCC/PT strategies to establish which type of strategy is more efficient across all load levels. The Window timer duration was evaluated by observing the mean freephone response delay when the SCP load is 0.8. As usual, the input freephone traffic scenarios described in Section 4.3.2 are used. In all cases, the same SCP and SSP service rates are used, to ensure a fair comparison between strategies. Also for all cases, the SCP load threshold is defined as being 0.8 Erlangs.

4.3.4.1 Stationary Behaviour

The input traffic offered in the stationary case is as shown in Figure 4.16. The resultant dynamic SCP load is as shown in Figure 4.37, while the mean SCP load is shown in Figure 4.38. Note that Window responds faster to the onset of overload than CCC/CG. This is due to the fact that it is always active and therefore responds immediately to the detection of overload, rather than having to wait, like CCC, until the end of the monitoring period. Window also exhibits fewer oscillations

in the dynamic SCP load than CCC/PT. This behaviour is similar to that of QLC (shown in Section 4.3.2), and is again based on the fact that it is always active. However, due to the fact that Window is based on a single fixed parameter (i.e. the query/response delay threshold used to set the Window timers), it does not provide premium SCP performance –it does not keep SCP load at the defined threshold. Here, for a low overload, the Window strategy overprotects the SCP, keeping the mean load at approximately 0.775. Also, Window does not converge as quickly as the dynamic CCC/PT, again as it is based on the use of a fixed delay parameter.

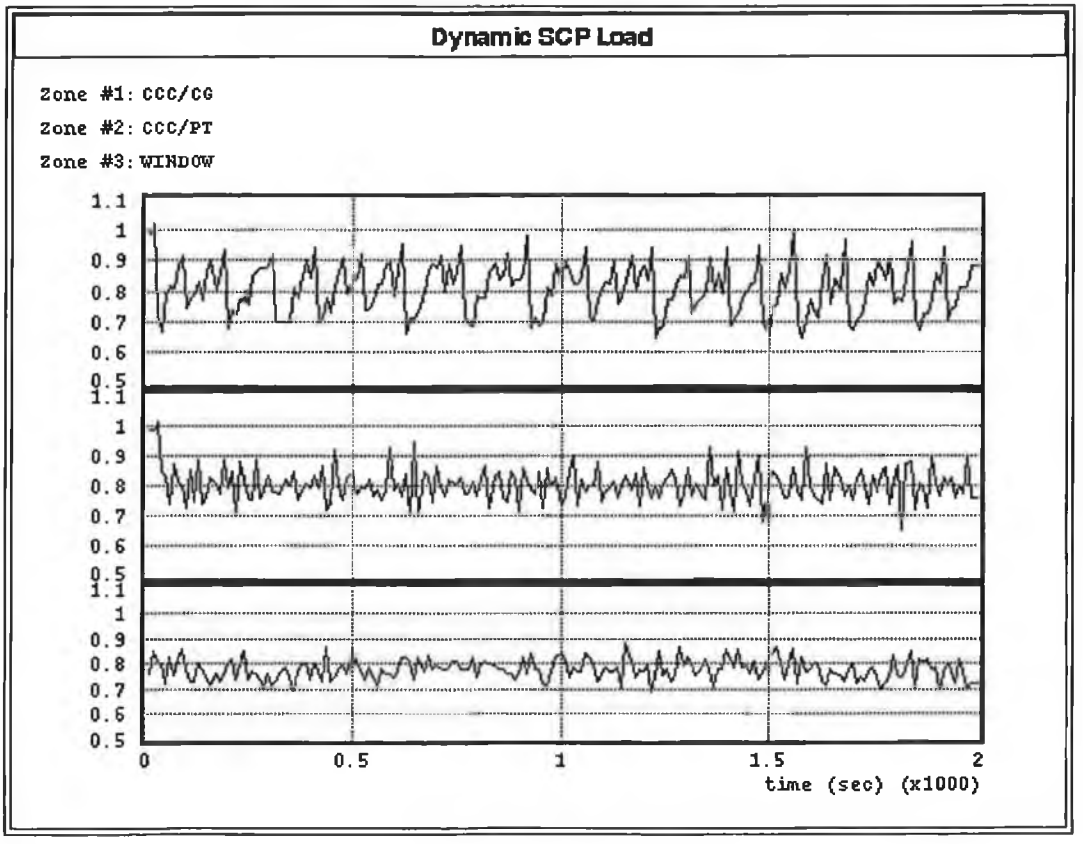


Fig. 4.37: Dynamic SCP load for stationary case

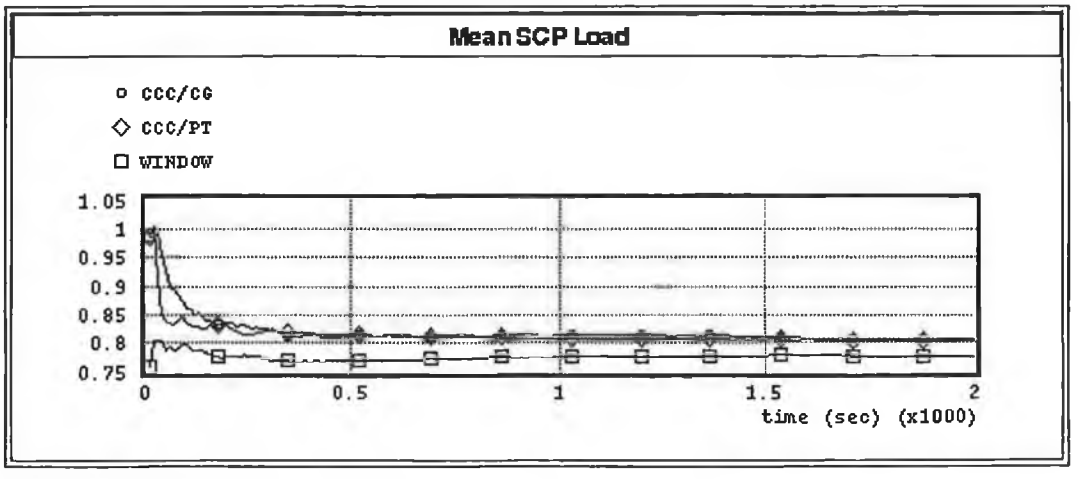


Fig. 4.38: Mean SCP load for stationary case

4.3.4.2 Behaviour under Linear Increase in Arrival Rates

To investigate the behaviour of each of the strategies under rapid increases in input traffic, as well as to establish their behaviour at low and high overload levels, the mean arrival rate of freephone traffic was increased linearly, as shown in Figure 4.19. The resultant mean SCP load is shown in Figure 4.39. Here, the disadvantages of basing a congestion control strategy on a single fixed parameter become apparent. At low overload levels (from 0.8 to 2.0 SCP Erlangs), the SCP is overprotected by Window, with load levels staying consistently below the SCP threshold. At high overload levels (above 2.0 Erlangs), the SCP is underprotected, with SCP load climbing as high as 0.93 Erlangs when applied load is over 4 Erlangs.

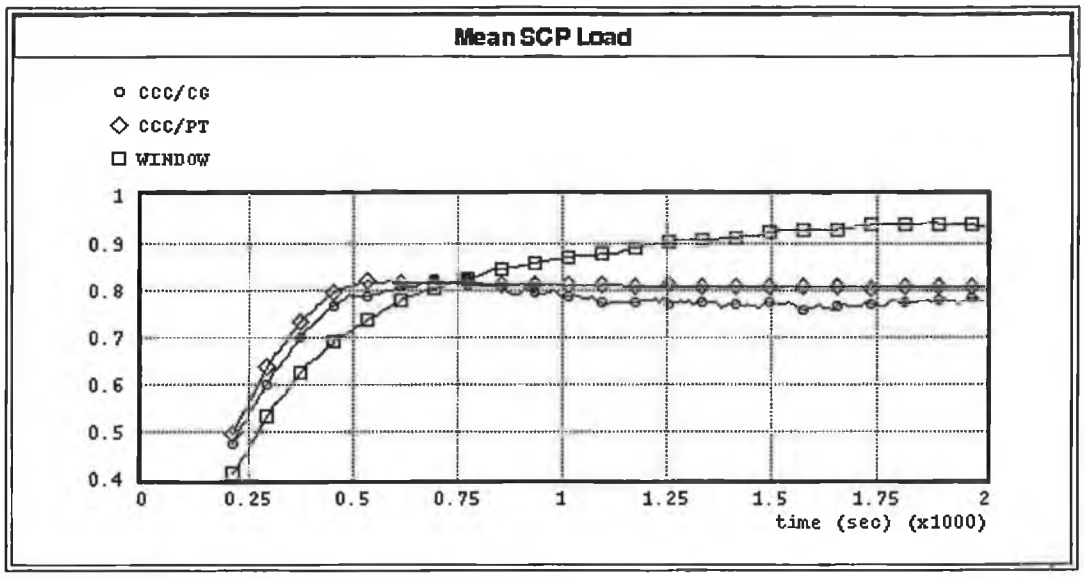


Fig. 4.39: Mean SCP load for linearly increasing arrival rates

4.3.4.3 Behaviour under Bursty Traffic Input

To investigate the behaviour of each of the methods under bursty traffic input, the arrival rates to the system were defined as shown in Figure 4.24. As expected, the results showed that Window, due to its active nature, is considerably faster to respond to the onset of congestion than either of the other two strategies (see Figure 4.40). Also, by strictly limiting access to the SCP, Window prevents the SCP from approaching saturation, unlike the CCC/CG and CCC/PT, both of which allow the SCP to become saturated for at least one monitoring interval. As such, Window provides by far the best reaction to bursty traffic.

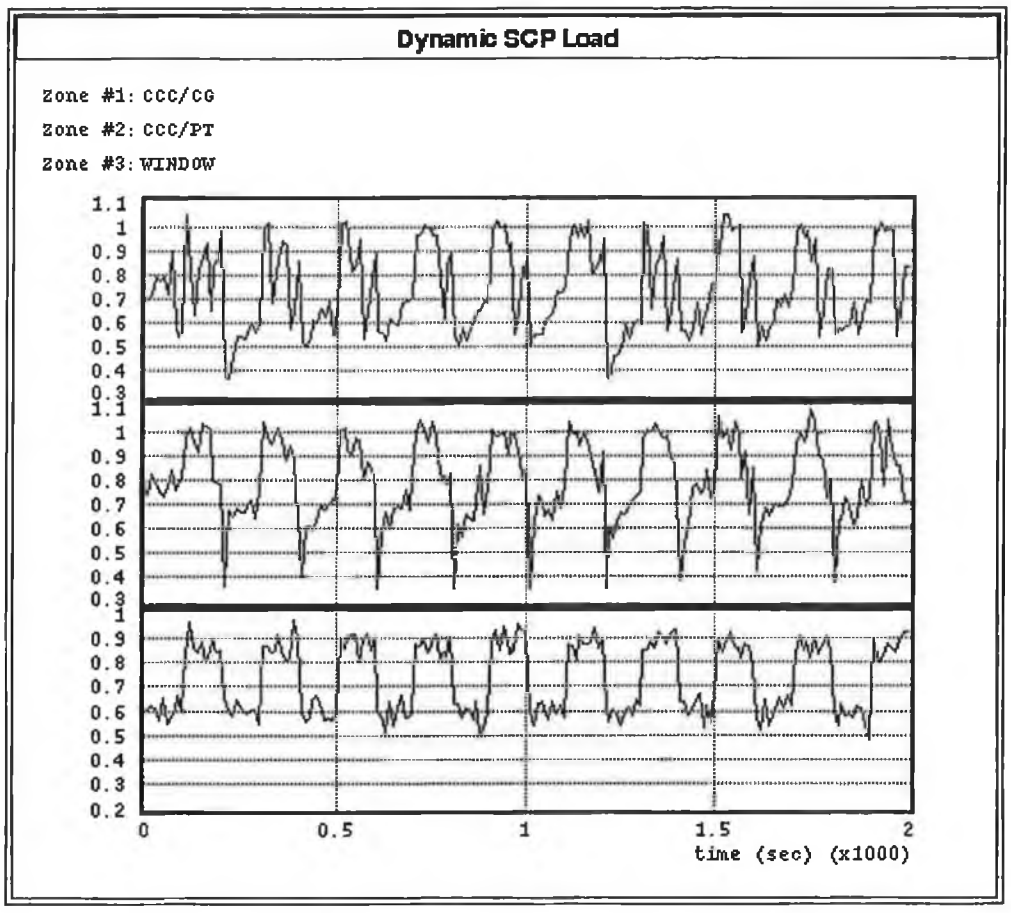


Fig. 4.40: Dynamic SCP load for bursty arrival rates

4.3.4.4 Summary of Results

The salient features of each of the strategies are outlined in Table 4.3 below, where a √ denotes acceptable behaviour and (*) denotes best behaviour for each category.

Category	*Window*	*Call Gapping*	*Percent Thinning*
Relative processor requirements	14.4	4.4	1
Rapid response to onset of congestion	√ (*)	√	
Rapid response to end of congestion	√ (*)	√	√
Speed of convergence	√		√
Consistency over all load levels		√	√ (*)

Table 2.3: Summary of Active vs Reactive Results

To summarise, while Window has the fastest reaction time, this is at the expense of considerable overheads. Window also fails to be effective over all ranges of overload, considerably overprotecting the SCP during low overload and underprotecting it during high overload. The conclusion is that CCC with a dynamic combination of CG and PT would probably provide consistently the best behaviour. This will be established in Chapter 6, where just such a strategy will be compared with Window.

4.4 Summary & Conclusion

There are a number of conclusions which may be drawn about the results as discovered in this chapter, the first of which relates to the main body of research completed - the investigation of existing congestion control strategies for the protection of the SCP of an Intelligent Network. The results acquired in Section 4.3.2 show that, while none of the existing detection methods provide satisfactory results over all load scenarios, CCC generally seems to perform better than the other strategies.

Of the throttles investigated, PT generally seems to outperform CG, with the exception of the most important characteristic – speed of response to overload. CG responds very quickly to overload, but its table-driven nature means that its response, while fast, is not very accurate and that it tends towards large oscillations and is very slow to converge to the defined threshold. PT, as a dynamic throttle, converges quickly and has the added advantage of being subscriber fair, in that it throttles all SSPs proportionally to their size.

Comparisons between CCC/PT, CCC/CG and Window provided inconclusive results. Window had the fastest response times to the onset of overload, but its behaviour is not consistent over all possible load levels. It also has considerably higher processor requirements than either of the two reactive communication-oriented strategies.

The principle drawback of virtually all the strategies investigated in this chapter is that they are table-driven, i.e. based on static parameters (PT being the only exception). This static nature has a number of implications. Firstly, configuring the algorithms by defining the best possible fixed parameters is extremely difficult. For strategies with a low number of parameters (e.g. Window), it is impossible to define parameters that will deal with all possible load levels correctly. Strategies with a large number of parameters (e.g. CCC, LMC, CG) tend to be able to handle larger variations in load, but also have a tendency to be extremely inaccurate –i.e. defining a number of parameters which will always cause the mean SCP load to converge to (or rather, to oscillate minimally around) the defined threshold for any offered load is not possible (in the experience of the author). A further difficulty with defining fixed parameter values is that as they are dependent

on the size of the resource at which they are located. As such, the parameters would have to be calculated independently for each SCP or SSP at which the strategies are targeted.

A second negative implication associated with static, table-driven detection strategies raises even greater concerns and cannot be resolved through the definition of the fixed parameters. This is the fact that any static detection algorithm implicitly makes assumptions about either the load requirements of the traffic types being managed or about the traffic mix. This issue was avoided in Section 4.3 by ensuring that all calls in the network were freephone calls. However, in reality, different services have very different characteristics. As an example, televoting call requests visit the SCP four times (rather than twice, like freephone) and, as such, require twice as much SCP processing as freephone calls. It would also take longer for the response to the initial televoting query to be returned to its source SSP than it does for the initial freephone response. Therefore, if the same parameters that were defined for freephone calls were used in a network that handles both freephone and televoting calls, none of the detection algorithms would respond correctly to the onset of congestion caused by televoting calls. This is shown in Figure 4.41, where overload occurs at t=200s, and may be described as follows:

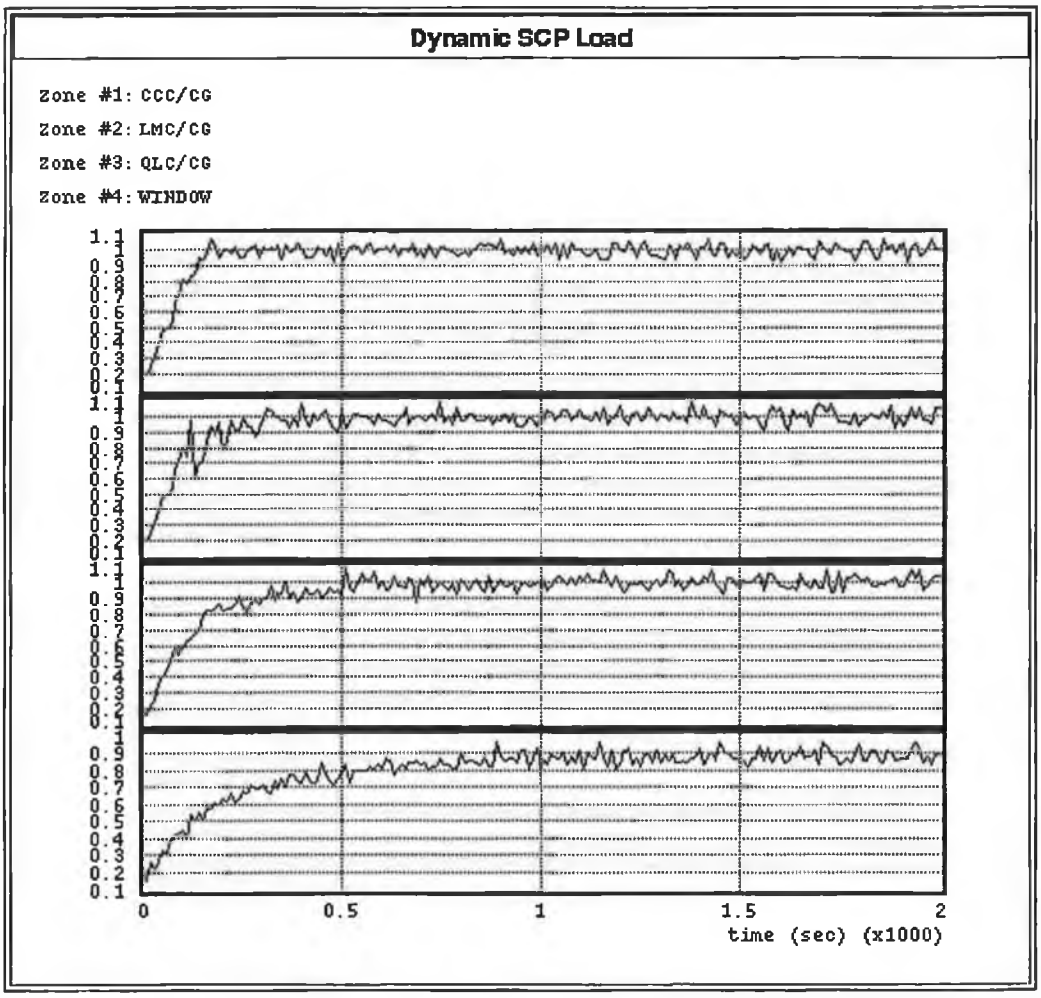


Fig. 4.41: Dynamic SCP load for televoting overload

- CCC would not detect overload until too late, as the number of arriving televoting calls that would cause overload is substantially less than for freephone. At the point where overload is finally detected, the SCP has already reached saturation.
- LMC responds very slowly to the detection of overload, as televoting has a load profile for which initial processing of requests at the SCP constitutes only 25% of their total load requirements. Therefore, by the time overload is detected, a large number of calls have already been accepted which must then complete processing at the SCP. Again, as with CCC, this leads to SCP saturation.
- QLC overreacts at low overload levels (even more so than in Section 4.3.2), as each televoting request must queue four times at the SCP. Then, as the level of overload grows, QLC becomes incapable of dealing with the rapidly rising queue lengths and the SCP becomes saturated.
- Window responds too quickly, detecting overload where none exists. This is due to the fact that the query/response delays for televoting are substantially higher than for freephone, so Window times out on televoting calls, even when no overload exists. However, as Window has a tendency to overprotect the SCP, its behaviour at high overload levels, while not ideal, is better than for the communication-oriented detection algorithms.

The conclusion of this is that no static detection algorithm will be able to protect the SCP efficiently when the input to the system is a varied mix of different call types with different load requirements and characteristics. This implies the need for a dynamic strategy, which will be able to calculate, based on the arriving traffic mix, the true state of overload of the SCP and react accordingly. A dynamic strategy would also have the added advantages (as shown by PT) of being easier to configure, as well being more accurate and converging to the threshold much faster than a table-driven static strategy.

As a final comment, existing models (including the one presented in this chapter) make the assumption that SSP resources are infinite, and therefore avoid all implications of possible SSP overload. In a real IN, this assumption is unreasonable. Therefore, to provide a real and comprehensive solution to IN congestion control, the implications of SSP overload and SSP protection must also be investigated and, if possible, a congestion control strategy should be developed which protects all IN resources from congestion.

Chapter 5

Global IN Congestion Control

5.1 Introduction

In the area of IN congestion control, it is recognised that, as the SCP is responsible for the execution of all services, it is crucial that its throughput is maximised at all times. Therefore, research to date has tended to focus on protecting the SCP from the effects of overload while maximising its efficiency at all times (see [Pham92], [Hebuterne90] and Chapter 4). The issue of SSP congestion was avoided in all work completed to date by raising the service rate of the SSP central processor (the CCF) to the point where congestion does not occur in the SSP. In reality, however, processing power at all SSPs will (obviously) be finite and therefore overload at SSPs also has the possibility of affecting IN performance. Also, a number of different scenarios exist for the implementation of INs. These include:

- An overlay network, where switches in the PSTN/ISDN route service requests to Service Switching Points (SSPs),
- The Service Node, where multiple IN physical elements are represented in one powerful node,
- An integrated network, where SSP functionality exists in the network switches.

These three implementation scenarios are depicted in Figure 5.1 below. The types of traffic routed through the IN SSPs therefore depend on the network implementation. In particular, in the integrated network implementation, SSPs will be required to process non-IN calls as well as IN calls. The long term view of IN evolution predicts that the integrated scenario will become more popular (e.g. with number portability looming, it may not be too long before the majority of calls are service related, and therefore it does not make sense for all of these calls to be routed to an overlay network or a service node. The more logical solution is to install service switching logic in all trunk switches). In this scenario, the issue of IN performance management widens in scope to include SPC performance management in multiple SSPs. In other words, it will no longer be sufficient to maximise SCP performance alone – it will be necessary to maximise the overall performance of the integrated network.

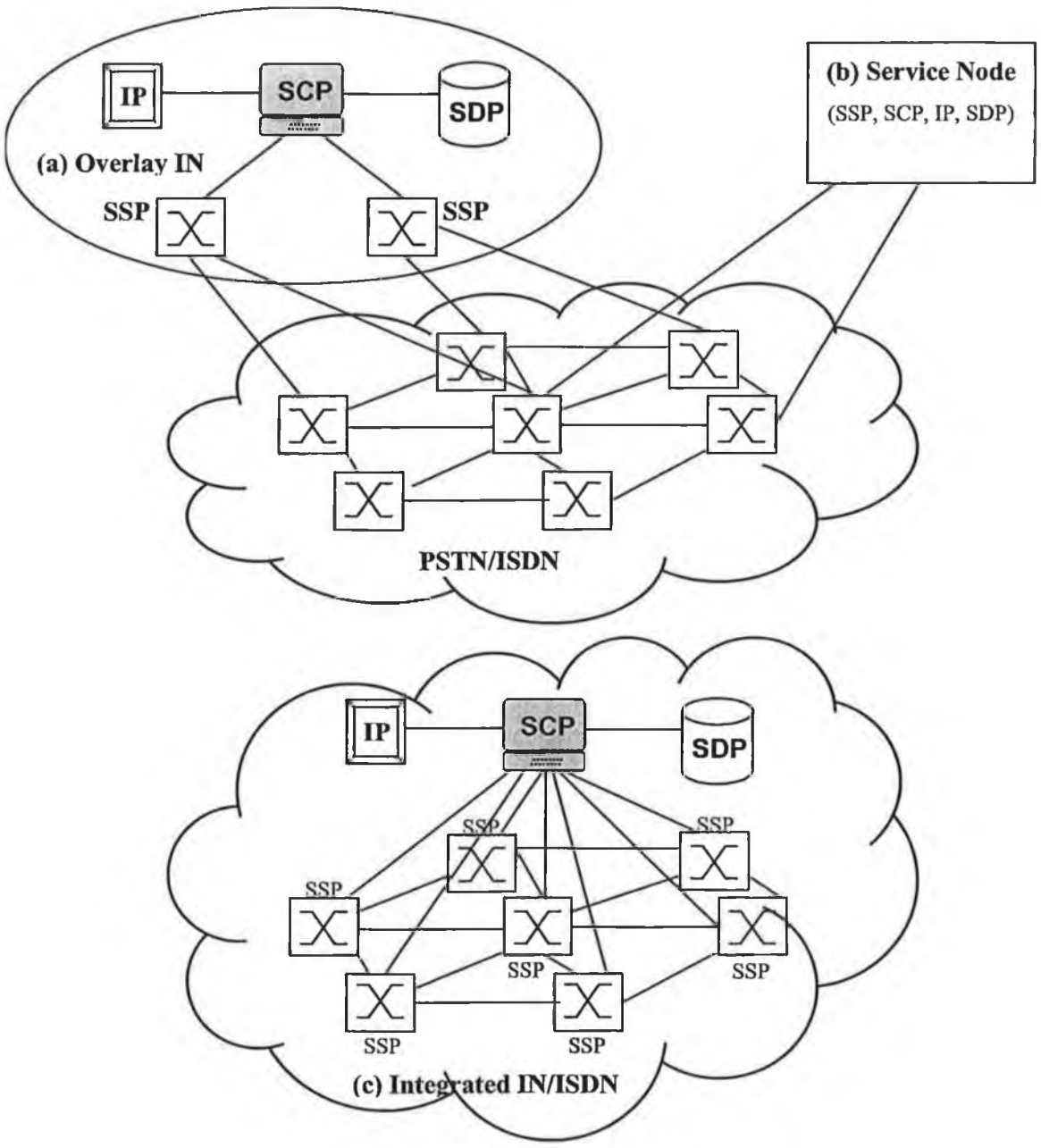


Fig. 5.1: IN Implementation Scenarios

At this point, based on the results of Chapter 4, let us also provide an enhanced definition of the requirements on an IN overload control strategy. Obviously, the basic requirements are as described in Chapter 2 - namely,

- Effectiveness: an overload control strategy must be able to protect IN resources under any load conditions and,
- Efficiency: the strategy must use processing resources in an efficient manner (in terms of keeping processing requirements as low as possible). Obviously, if it is desirable to incorporate e.g. priorities in a congestion control scheme, the added complexity of the

algorithm will cause it to have a larger footprint. This is only justifiable if the benefits of the complex scheme outweigh the processor usage costs.

However, there are also other highly desirable characteristics of an overload control strategy, which include:

- Scalability: the algorithm should not be dependent on the size of its resource or the mix or arrival rates of applied traffic, i.e. it should not be dependent on any explicit fixed parameters. This would ensure that the algorithm is both simple to install at any resource and that it reacts correctly and accurately to the applied traffic.
- Flexibility: a well-designed algorithm should be easily customised to include factors such as different call priorities and load requirements or different requirements on fairness.
- Fairness: There are two main interpretations of fairness in the IN context, as described in Chapter 2, section 2.3.1. "Service fairness" means that if overload is caused by an excess of calls of one particular service type, only calls of that type should be rejected (i.e. Focussed Destination Overload Control - FDOC). "Subscriber fairness" means that the probability of rejection ought to be the same for all the subscribers of a particular service, irrespective of which SSP they are connected. Ideally, an IN congestion control algorithm should exhibit both types of fairness.

Many of the IN overload control algorithms which have been proposed in the past and discussed in Chapter 4 go some way towards meeting the basic requirements on an IN overload control strategy. However, these basic algorithms tend not to have the desirable characteristics described above. A number of strategies have also been proposed which do address various of these characteristics in the area of SCP protection (e.g. [Rumsewicz95], [Lee97], [Smith95] and [Lodge98b]) and so reach greater SCP performance efficiency, where the criteria generally used to evaluate the performance efficiency of an overload control algorithm includes (but is not restricted to) SCP load, SCP queue length, the number of IN calls rejected, cost efficiency of the algorithm and mean service delays.

The aim of this work is to find an overload control strategy which encapsulates both the basic requirements and all desired characteristics, in the global performance management of an IN which consists of one SCP and multiple SSPs and which handles a number of different traffic types (IN and non-IN) with different call characteristics fairly. As an example of service unfairness, all currently defined strategies are specifically designed to protect just the SCP of the IN and therefore, in ensuring maximum possible SCP efficiency, the performance of non-IN processing in the SSPs may be degraded. This should not be allowed to happen as, although some service related calls may provide more revenue to network operators, the ratio of revenue to resource requirements (let's call it the Rev/Res ratio) for non-IN calls may be greater and therefore, to

maximise the performance of the overall network, non-IN calls should not have to receive a degraded quality of service during times when service related traffic is too high.

Examining the area of global IN performance management, we see that, in the past, the problem of switch congestion control was much simpler to address than it will be in an integrated IN environment. Congestion controls for the protection of SSPs have either been unnecessary (e.g. in a service node scenario) or based on standard SPC control mechanisms. However, when SSPs must deal with multiple call types, including non-IN and IN, all of which have different priorities, revenues and call load distribution curves (as opposed to ISDN calls which have the same load distribution curve, as shown in [Seraj85], [Hubig94] and Chapter 2), it will be neither efficient nor fair to throttle all calls equally upon occurrence of overload. Also, if the congestion control strategies at the SCP and SSP work independently of each other (i.e. each element is responsible for its own protection only), conflicts between SCP and SSP strategies could result in lower overall network efficiency.

To estimate the impact of these types of conflicts on overall network performance and to prove that a global IN congestion control strategy is essential, the model of the IN described in Chapter 4 was extended to include multiple finite-capacity SSPs, each of which processes both IN and non-IN call types. This new model (in both simulation and analytic form) is described in Section 5.2 below. Two different control scenarios were then put in place on the model. For the first scenario, independent control strategies were put in place at the SCP and SSPs. For the second scenario, a simple strategy was devised to throttle incoming traffic at each SSP based on the load levels at both the SCP and SSP. This experiment and its conclusions are described in Section 5.3. Section 5.4 presents a new global IN congestion control strategy based on revenue optimisation, while Section 5.5 gives results for this strategy.

5.2 The New, Comprehensive IN Models

The models used in Chapter 4 are sufficient only for estimating the efficiency of SCP congestion controls and for establishing the necessity for throttling IN calls at the SSP. To facilitate the investigation of congestion control strategies for dealing with all types of calls in situations of both SSP and/or SCP congestion, a new, more detailed design model was developed, and implemented both as a simulation model (in OPNET) and as an analytic model (to facilitate mathematical analysis of the behaviour of the system).

5.2.1 The IN Model Design

The new model design of the SSP was developed in order to provide an accurate representation of both switching functionality and service handling procedures. This model is shown in Figure 5.1 and closely represents the operation of the central controller of the Ericsson switch and is similar to that described in [Seraj85]. The resource to be maximised is the capacity of this central processor as all intelligence resides here. The similarities between the operation of this model and that of the Ericsson switch is deliberate, with the aim that all results gained from this model may be directly applicable as a real congestion solution in a real system.

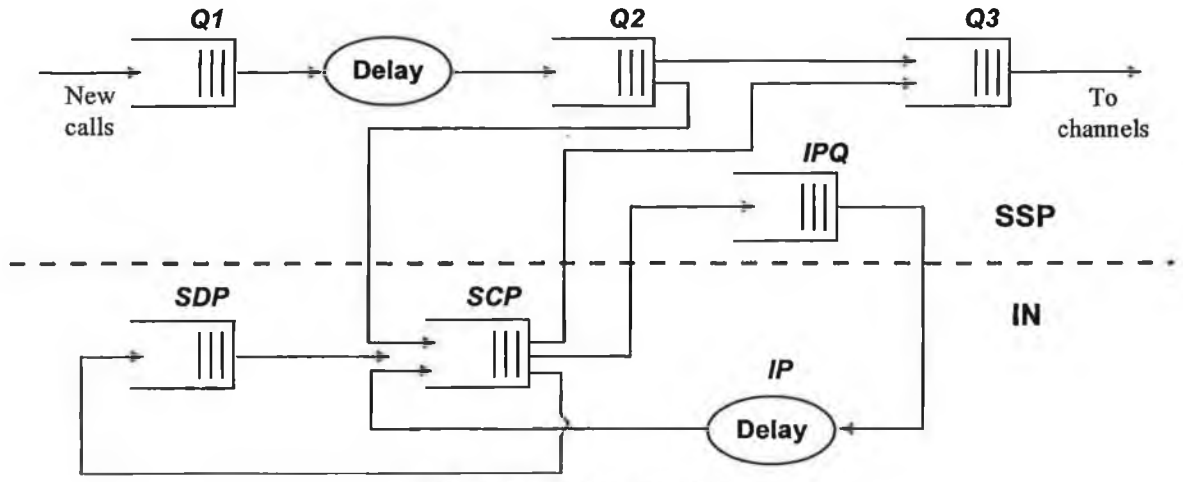


Fig. 5.2: The IN model design

Operation of this model may be explained as follows: all new call requests arrive into Q1 in the central processor. If the central controller has capacity available and the user has the authority to make a call, call acceptance is returned to the user in the form of a dial tone. Once the user has been supplied with a dial tone, any digits entered are collected and returned to Q2 in the central controller. At this buffer, service related calls may be differentiated from ordinary calls. If the call is a non-IN call, the request is forwarded to Q3 for routing, while IN calls are sent to the SCP. Any database access requirements are fulfilled through the exchange of messages between the SCP and SDP. User interaction requests are sent to the IPQ in the SSP, the service time of which represents the time required to open a channel between the user and the IP - note that this queue is included in the SSP side of the model, as establishing the channel will require some SSP central processor capacity. After the channel has been opened, the request is passed to the IP, where it is delayed (representing interaction with the user) before returning to the SCP. When service execution has completed at the SCP, calls are forwarded to Q3 for further routing (e.g. in the case of freephone calls) or termination (e.g. in the case of televoting calls).

Current rules for overload control state that, if a call is to be rejected, it should be as soon as possible in call processing, for two reasons:

1. To minimise the wastage of central processor capacity,
2. To minimise delays to users.

According to these rules, it is possible to reject calls at both Q1 and Q2, but not at Q3. Note that, as it is not possible to distinguish between call types in Q1, throttling of calls at this queue can only be applied to all call types equally. Differentiation between call types occurs during processing at Q2 - it is therefore possible to selectively restrict different call types at this queue. However, all calls that are accepted at Q3 must be given the chance to complete successfully - i.e. they cannot be rejected for any reason. Therefore, all calls placed in buffer Q3 are guaranteed the processing time they require to complete successfully. Therefore, the only buffers at which calls may be rejected are Q1 (unconditional rejection to protect the SSP) and Q2 (conditional rejection based on call type). In terms of devising a flexible overload control algorithm, it is far more desirable to place an overload control strategy at Q2, as this gives scope for selective throttling of calls based on various criteria (e.g. fairness, load requirements, priorities etc.).

All central controller buffers (Q1, Q2, Q3 and the IPQ) are served by a single processor. Current priorities in the Ericsson switch define that Q3 is provided with the highest servicing priority to ensure that calls which have not been rejected complete successfully. Q2 is has the next highest priority, as all calls there have already received some processing time in Q1 and should therefore be given a good opportunity to complete. Q1 has the lowest priority and only receives processing if sufficient capacity is available.

In the design, processor capacity is allocated to each queue on a priority basis similar to the AXE priorities defined above, but amended to include the IPQ. Therefore, highest priority is assigned to the IPQ, lowest priority assigned to Q1 and equivalent priorities assigned to Q2 and Q3. Q3 has a single service rate, while Q1 has two service rates - one for processing accepted traffic and the other for rejecting calls. Q2, on the other hand, has three different service rates – the first for accepting non-IN calls, the second for accepting IN calls and the third for the rejection of calls. The service times of these queues are set to reflect the load distribution of a non-IN call - i.e. mean Q1 service time (representing call authorisation) is shorter than that of Q3 or the mean non-IN acceptance time at Q2, while the mean non-IN acceptance time at Q2 (representing non-IN number analysis) and Q3 (routing) are equivalent. The IN acceptance time at Q2 is set as a factor of 2.5 greater than the acceptance time for non-IN calls – this is an approximation of the excess processing requirements of IN calls over non-IN calls in a real SSP. Further, rejection rates at both Q1 and Q2 are defined as being much greater than their acceptance rates. As a result of these defined service rates, any overload will cause congestion at Q2 prior to affecting any other SSP queue. Therefore, any SSP congestion detection algorithm should be located at Q2.

Five types of call are defined for the model - these are international, international freephone, televoting, local and freephone. Table 5.1 shows the identifier assigned to each call type, along with the set route that is followed by the call type through the system.

Call Type	*Identifier*	*Route through IN from SSP n*
International (non-IN)	1	$Q1_n$-$Q2_n$-$Q3_n$
International Freephone (IN)	2	$Q1_n$-$Q2_n$-SCP-SDP-SCP-$Q3_n$
Televoting (IN)	3	$Q1_n$-$Q2_n$-SCP-IPQ_n-SCP-SDP-SCP-IPQ_n-SCP-$Q3_n$
Local (non-IN)	4	$Q1_n$-$Q2_n$-$Q3_n$
Freephone (IN)	5	$Q1_n$-$Q2_n$-SCP-SDP-SCP-$Q3_n$

Table 5.1: Calls types in enhanced IN model

Retrials of 30% are also included in the model, i.e. 30% of all calls rejected will be retried in the following interval (a simplification of the assumptions outlined in [Manfield91]). All call type arrival rates are Poisson.

5.2.2 The IN Simulation Model

The existing simulation model in OPNET was enhanced to reflect the new SSP structure, and was further extended to contain five such SSPs, with call types as shown in Table 5.1. The new network layer model in OPNET is depicted in Figure 5.3. As shown, the model consisted of five SSPs, one SCP and an SDP.

As in the IN simulation model used in Chapter 4, the SDP has a deterministic service time distribution, while the SCP has an exponential service time distribution. The only change made to the **scp** node model was the inclusion of an SLP for international freephone. All other changes to the OPNET model used in Chapter 4 took place in the model of the **IN_ssp** node. These changes will be described next.

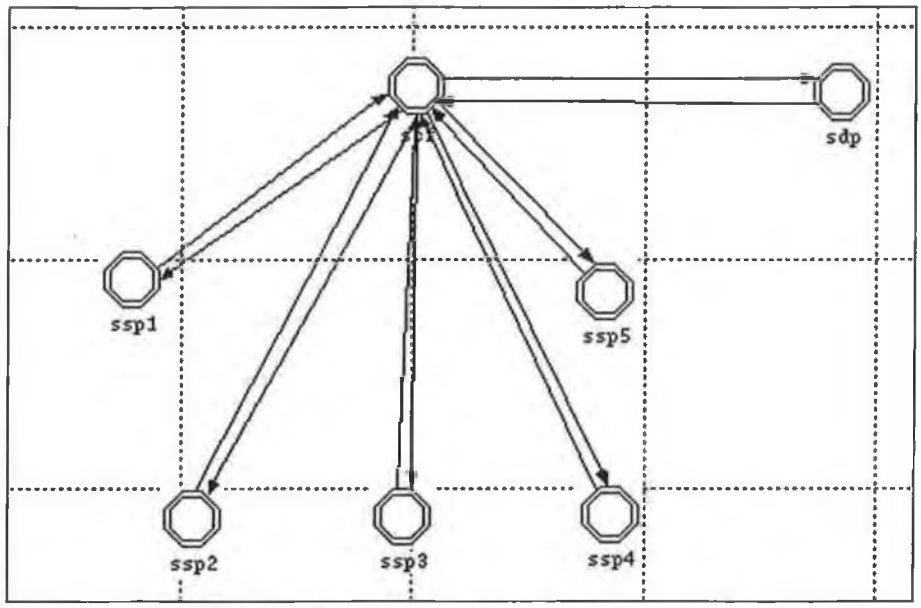


Fig. 5.3: The New IN Simulation Model (Network Layer)

5.2.2.1 The IN_ssp Node Model

The operation of the SSP was amended to reflect the new SSP design, as shown in Figure 5.4. The primary changes that took place in the node model described in Chapter 4 include:

- The *ccf* process was replaced by three processes, representing Q1, Q2 and Q3. All the service times of each of these new queues (i.e. acceptance and rejection rates, as described in Section 5.2.1) were set to be exponentially distributed.
- The throttles that were implemented in the old ccf are now in Q1 and Q2. The Q2 throttles were also extended so that they can be applied either to all services (including non-IN services) or to individual service types. This is to facilitate selective throttling.
- The termination of non-IN calls is no longer included in the SSP. This is because all conflict for processor resources in the SSP takes place during the Call Authorisation and Number Translation states of the Originating BCSM. If a call requests is accepted in both of these states, then it must complete successfully and therefore it is unnecessary to represent the processing allocated to the states further on in the call state model.
- The SRF is modelled differently. In Chapter 4, this was modelled as an Erlang-C queue. For this work, however, the behaviour of the IP that is of primary interest is the processing requirements it places on the SSP to establish a channel between the user and itself. Therefore, we replace the SRF with a queue (the *ipq*) to model the processing of these requests for a channel at the SSP. This new *ipq* is defined as having an exponentially distributed service time, representing the time it takes for the SSP to establish the requested channel. This is then followed by a simple uniformly distributed delay around a mean of 5 seconds.

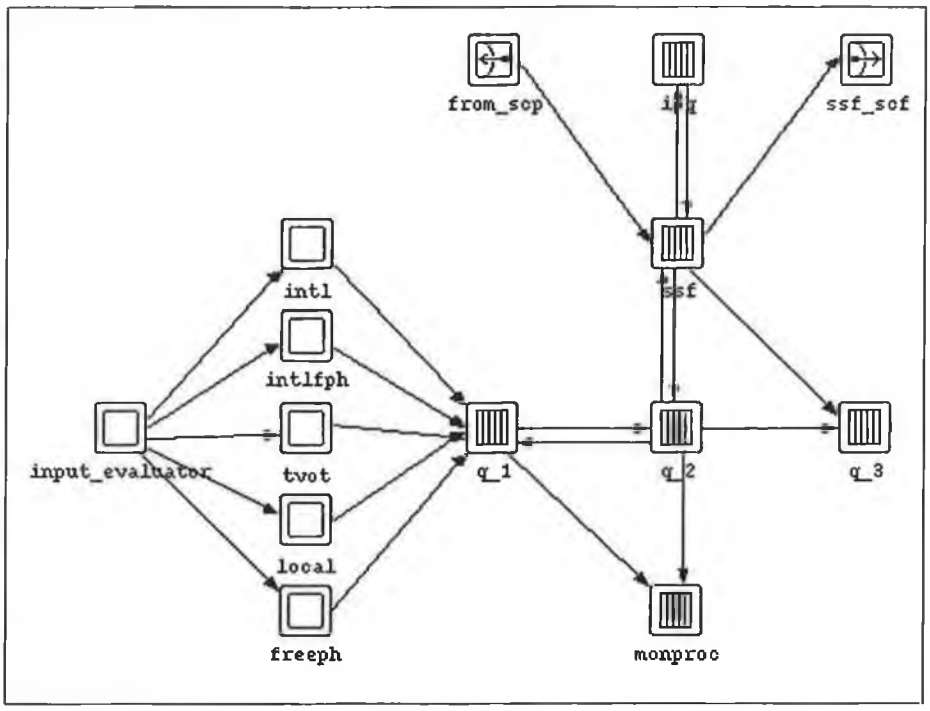


Fig. 5.4: The New IN_ssp Node Model

5.2.3 The IN Analytic Model

The queuing model has one SCP, one SDP and multiple SSPs, as shown in Figure 5.2. The behaviour of the system is almost identical to that described for the simulation model in section 5.2.2. The primary differences between the analytic model and the simulation model of the IN are as follows:

- The uniformly distributed delay representing IP interaction with the user is omitted from the analytic model as, in steady state, the time spent interacting with the user has no effect on system performance.
- For ease of analysis, all queues in the systems – including the SDP - have exponentially distributed service time distributions.
- Percent thinning is the only throttling mechanism available at Q1 and Q2, as CG cannot be modelled accurately analytically. Therefore, in investigations where CG is used in the simulation model, some small discrepancies between the simulation and analytical results are to be expected.

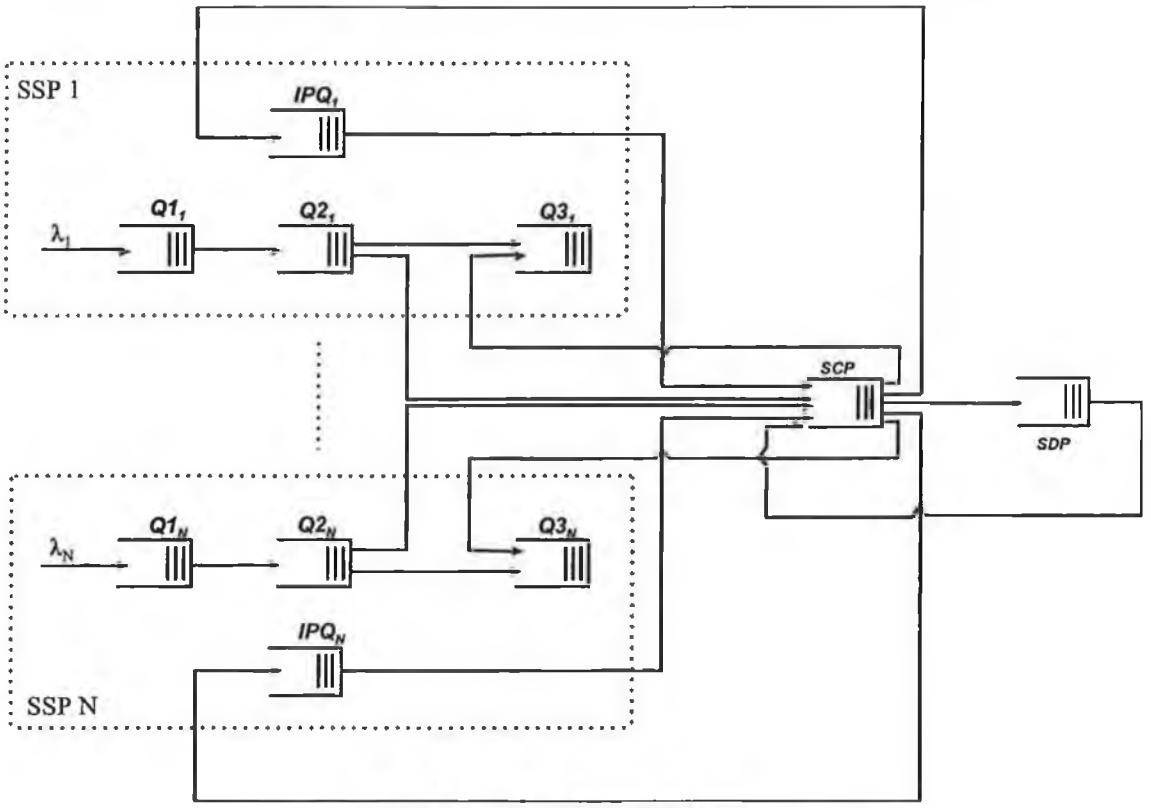


Fig. 5.5: The IN Analytical Model

The model contains the five call types defined in Table 5.1. Any throttle at Q1 can only reject all calls equally, and we therefore define P_n^a as the Q1 global percent thinning throttling argument for SSP n. We further define one Q2 percent thinning throttling argument for each call type - let us define $p^a{}_{n,i}$ as the percentage of offered calls of type i to be accepted at Q2 of SSP n (during the next interval). Therefore, we have five probabilities of acceptance at Q2, namely $p^a{}_{n,1},..,p^a{}_{n,5}$. For completeness, retrials are included, where we choose that 30% of call attempts rejected in one interval are retried in the next interval, i.e. $\lambda^r_{n,i}(t) = (0.3)(1 - P_n^a(t-T)p^a{}_{n,i}(t-T))\lambda^o_{n,i}(t-T)$ is the retrial arrival rate of call type i at SSP n, where $\lambda^o_{n,i}$ is the original arrival rate of call type i at SSP n and T is the interval length. Therefore, the total (Poisson) arrival rate at $Q1_n$ is $\lambda_n(t) = \sum_i \lambda_{n,i}(t) = \sum_i (\lambda^o_{n,i}(t) + \lambda^r_{n,i}(t))$.

To simplify network analysis, $Q3_n$, the SCP, SDP and IPQ_n each have a single service time with a mean of, respectively, $\mu^{-1}_{Q3_n}, \mu^{-1}_{SCP}, \mu^{-1}_{SDP}$ and $\mu^{-1}_{IPQ_n}$. The throughputs of these queues are $\rho_{Q3_n}, \rho_{SCP}, \rho_{SDP}$ and ρ_{IPQ_n}. On the other hand, $Q1_n$ has two service times defined, with call rejection time much lower than processing time for accepted calls. $Q1_n$ service rates are therefore $\mu_{Q1_n,acc}$ and $\mu_{Q1_n,rej}$. Its resultant throughput is ρ_{Q1_n}. $Q2_n$ has three service times defined, call

rejection time, non-IN acceptance time and IN acceptance time. $Q2_n$ service rates are therefore $\mu_{Q2_n,rej}$, $\mu_{Q2_n,acc,non}$ and $\mu_{Q2_n,acc,IN}$. The throughput of $Q2_n$ is ρ_{Q2_n}. Note that at this time, for the purposes of generalisation, we make no assumptions as to the service time distributions used. Therefore, the only assumption made at this point is that all arrival processes to the system are Poisson.

5.2.3.1 Estimating Arrival Rates and Loads for the IN Analytic Model

Using our knowledge of the routes taken by each call type through the queuing system, we may specify the mean arrival rates at each queue in the model:

For $Q1_n$: $\lambda_{Q1_n}(t) = \lambda_n(t) = \sum_{i=1}^{5} \lambda_{n,i}(t)$

For $Q2_n$: $\lambda_{Q2_n}(t) = P_n^a \lambda_{Q1_n}(t) = P_n^a \sum_{i=1}^{5} \lambda_{n,i}(t)$

For the SCP : $\lambda_{SCP}(t) = \sum_{j=2,3,5} e_{SCP,j} \sum_{n=1}^{N} P_n^a p_{n,j}^a \lambda_{n,j}(t)$

For the SDP : $\lambda_{SDP}(t) = \sum_{j=2,3,5} e_{SDP,j} \sum_{n=1}^{N} P_n^a p_{n,j}^a \lambda_{n,j}(t)$ where $e_{SDP,j} = 1$ for $j = 2,3,5$

For IPQ_n : $\lambda_{IPQ_n}(t) = e_{IPQ,3} P_n^a p_{n,3}^a \lambda_{n,3}(t)$ where $e_{IPQ,3} = 2$

For $Q3_n$: $\lambda_{Q3_n}(t) = P_n^a \sum_{i=1}^{5} p_{n,i}^a \lambda_{n,i}(t)$

where $e_{SCP,j}$ is the number of times a call request of type j enters the SCP during its execution, $e_{SDP,j}$ is the number of times a request of type j will receive processing at the SDP and $e_{IPQ,j}$ is the number of times a call of type j will pass through the IPQ. In a similar manner, we may define the loads at each queue in the model:

For $Q1_n$: $\rho_{Q1_n}(t) = \frac{\lambda_n P_n^a}{\mu_{Q1_n,acc}} + \frac{(1-\lambda_n P_n^a)}{\mu_{Q1_n,rej}}$

For $Q2_n$: $\rho_{Q2_n}(t) = \sum_{i=1}^{5} P_n^a \lambda_{n,i} \left(\frac{p_{n,i}^a}{\mu_{Q2_n,acc,i}} + \frac{(1-p_{n,i}^a)}{\mu_{Q2_n,rej}}\right)$

For the SCP : $\rho_{SCP}(t) = \frac{\lambda_{SCP}(t)}{\mu_{SCP}}$

For IPQ_n : $\rho_{IPQ_n}(t) = \frac{\lambda_{IPQ_n}(t)}{\mu_{IPQ_n}}$

For the SDP : $\rho_{SDP}(t) = \frac{\lambda_{SDP}(t)}{\mu_{SDP}}$

For $Q3_n$: $\rho_{Q3_n}(t) = \frac{\lambda_{Q3_n}(t)}{\mu_{Q3_n}}$

5.2.3.2 Estimating Service Delays for the IN Analytic Model

The decomposition method (described in Chapter 3) is used to estimate service delays in the analytic model, as it provides a good approximation of multiqueue systems in the case when the user wants to estimate or control mean queue lengths and/or mean delays. The following equations were generated using the decomposition method in the form described in Chapter 3, where (to simplify the notation) all parameters on the right hand side are as defined for time interval $[t\text{-}T,t]$, with the exception of P_n^a and $p_{n,i}^a$, which are as defined at time $(t\text{-}T)$:

For $Q1_n$: $Ka_{Q1_n}(t)=1$ (Poisson arrival rates)

$$C_{Q1_n}(t)=1-2\rho_{Q1_n}{}^2+\lambda_n^2\left(\frac{P_n^a(Ks_{Q1_n,acc}+1)}{\mu^2_{Q1_n,acc}}+\frac{(1-P_n^a)(Ks_{Q1_n,rej}+1)}{\mu^2_{Q1_n,rej}}\right)$$

For $Q2_n$: $Ka_{Q2_n}(t)=C_{Q1_n}(t)$

$$C_{Q2_n}(t)=Ka_{Q2_n}(t)(1-\rho_{Q2_n})+\rho_{Q2_n}(1-2\rho_{Q2_n})+P_n^a\lambda_n\sum_i\lambda_{n,i}\left(\frac{p_{n,i}^a(Ks_{Q2_n,acc,i}+1)}{\mu^2_{Q2_n,acc,i}}+\frac{(1-p_{n,i}^a)(Ks_{Q2_n,rej}+1)}{\mu^2_{Q2_n,rej}}\right)$$

For $Q3_n$:

$$Ka_{Q3_n}(t)=\frac{1}{\lambda_{Q3_n}}\left[\left[(C_{Q2_n}(t)-1)\left(\frac{1}{\lambda_n}\sum_{k=1,4}p_{n,k}^a\lambda_{n,k}\right)+1\right]P_n^a\left(\sum_{k=1,4}p_{n,k}^a\lambda_{n,k}\right)+\left[(C_{SCP}(t)-1)\left(\frac{P_n^a}{\lambda_{SCP}}\sum_{j=2,3,5}p_{n,j}^a\lambda_{n,j}\right)+1\right]P_n^a\left(\sum_{j=2,3,5}p_{n,j}^a\lambda_{n,j}\right)\right]$$

$$C_{Q3_n}(t)=\rho_{Q3_n}+Ka_{Q3_n}(t)(1-\rho_{Q3_n})+\rho^2_{Q3_n}(Ks_{Q3_n}-1)$$

For the SCP:

$$Ka_{SCP}(t)=\frac{1}{\lambda_{SCP}}\left[\sum_{n=1}^{N}\left[(C_{Q2_n}(t)-1)\left(\frac{1}{\lambda_n}\sum_{j=2,3,5}p_{n,j}^a\lambda_{n,j}\right)+1\right]P_n^a\left(\sum_{j=2,3,5}p_{n,j}^a\lambda_{n,j}\right)+C_{SDP}(t)\lambda_{SDP}+\sum_{n=1}^{N}C_{IPQ_n}(t)\lambda_{IPQ_n}\right]$$

$$C_{SCP}(t)=\rho_{SCP}+Ka_{SCP}(t)(1-\rho_{SCP})+\rho^2_{SCP}(Ks_{SCP}-1)$$

For the SDP:

$$Ka_{SDP}(t)=1+(C_{SCP}(t)-1)\left(\frac{1}{\lambda_{SCP}}\sum_{n=1}^{N}P_n^a\sum_{j=2,3,5}p_{n,j}^a\lambda_{n,j}\right)$$

$$C_{SDP}(t)=\rho_{SDP}+Ka_{SDP}(t)(1-\rho_{SDP})+\rho^2_{SDP}(Ks_{SDP}-1)$$

For IPQ_n:

$$Ka_{IPQ_n}(t)=1+(C_{SCP}(t)-1)\left(\frac{2P_n^a p_{n,3}^a\lambda_{n,3}}{\lambda_{SCP}}\right)$$

$$C_{IPQ_n}(t)=\rho_{IPQ_n}+Ka_{IPQ_n}(t)(1-\rho_{IPQ_n})+\rho^2_{IPQ_n}(Ks_{IPQ_n}-1)$$

where Ka_X is the square of the variation coefficient (svc) of interarrivals at queue X, Ks_X is the svc of the service time at queue X and C_X is the svc of the intervals between two successive departures from queue X. Knowing the arrivals rates and service rates for interval $[t\text{-}T,t]$ and the

acceptance probabilities as defined at time (t-T), the decomposition equations may be solved to find the value of Ka_X for each queue in the system. Then, again as described in Chapter 3, Kingman's formula will allow the mean queue length of each queue to be calculated, according to:

$$\overline{N}_X = \rho_X\left(1+\frac{\rho_X(Ka_X+1)}{2(1-\rho_X)}\right)$$

The average response time for each queue may then be evaluated using Little's Law ($\overline{T}_X = \overline{N}_X / \lambda_X$), and the average response time for each service type generated at each SSP can be estimated by summing the delays at each queue along its route.

5.3 Estimation of the Effects of non-IN Traffic and Finite SSP Resources on IN Performance

This work was carried out using the models defined in Section 5.2. Two assumptions were made: that all service times are exponentially distributed, and that there is only one SSP (no more are needed as this investigation seeks to prove only that a problem exists). This results in the specialisation of the general decomposition equations of Section 5.2.3.2, as shown below. Note that again, all parameters on the right hand side are as defined for time interval [t-T,t], with the exception of P_n^a and $p_{n,i}^a$, which are as defined at time (t-T).

For Q1: $Ka_{Q1}(t)=1$ (Poisson arrival rates)

$$C_{Q1}(t)=1-2\rho_{Q1}{}^2+2\lambda^2\left(\frac{P^a}{\mu^2_{Q1,acc}}+\frac{(1-P^a)}{\mu^2_{Q1,rej}}\right)$$

For Q2: $Ka_{Q2}(t)=C_{Q1}(t)$

$$C_{Q2}(t)=Ka_{Q2}(t)(1-\rho_{Q2})+\rho_{Q2}(1-2\rho_{Q2})+2P^a\lambda\sum_i\lambda_i\left(\frac{p_i^a}{\mu^2_{Q2,acc,i}}+\frac{(1-p_{n,i}^a)}{\mu^2_{Q2,rej}}\right)$$

For Q3:

$$Ka_{Q3}(t)=\frac{1}{\lambda_{Q3}}\left[\left[(C_{Q2}(t)-1)\left(\frac{1}{\lambda}\sum_{k=1,4}p_k^a\lambda_k\right)+1\right]P^a\left(\sum_{k=1,4}p_k^a\lambda_k\right)+\left[(C_{SCP}(t)-1)\left(\frac{P_n^a}{\lambda_{SCP}}\sum_{j=2,3,5}p_j^a\lambda_j\right)+1\right]P^a\left(\sum_{j=2,3,5}p_j^a\lambda_j\right)\right]$$

$$C_{Q3}(t)=\rho_{Q3}+Ka_{Q3}(t)(1-\rho_{Q3})$$

For the SCP:

$$Ka_{SCP}(t)=\frac{1}{\lambda_{SCP}}\left[\left[(C_{Q2}(t)-1)\left(\frac{1}{\lambda}\sum_{j=2,3,5}p_j^a\lambda_j\right)+1\right]P^a\left(\sum_{j=2,3,5}p_j^a\lambda_j\right)+C_{SDP}(t)\lambda_{SDP}+C_{IPQ}(t)\lambda_{IPQ}\right]$$

$$C_{SCP}(t)=\rho_{SCP}+Ka_{SCP}(t)(1-\rho_{SCP})$$

For the SDP:

$$Ka_{SDP}(t)=1+(C_{SCP}(t)-1)\left(\frac{P^a}{\lambda_{SCP}}\sum_{j=2,3,5}p_j^a\lambda_j\right)$$

$$C_{SDP}(t)=\rho_{SDP}+Ka_{SDP}(t)(1-\rho_{SDP})$$

For the IPQ: $Ka_{IPQ}(t) = 1 + (C_{SCP}(t) - 1)(\frac{2P^2 p_3^3 \lambda_3}{\lambda_{SCP}})$

$$C_{IPQ}(t) = \rho_{IPQ} + Ka_{IPQ}(t)(1 - \rho_{IPQ})$$

5.3.1 Strategies used for Comparison

Two strategies were implemented on both the IN simulation and analytic models. As the analytic model represents the ideal case, the simulated and analytic models should exhibit similar behaviour, i.e. the analytic model should return the mean of the simulated model results. For the first scenario implemented, independent control strategies were put in place at the SCP and SSPs. For the second scenario, a simple strategy was devised to throttle incoming traffic at the SSP based on the load levels at both the SCP and SSP, i.e. this strategy controlled the SSP and SCP traffic jointly. An assumption made for this study is that IN calls have a higher priority than non-IN calls and therefore, where possible, the number of IN calls accepted should be maximised.

5.3.1.1 The Independent IN Congestion Control Strategy

For the independent strategy, the SCP congestion control strategy was based on the use of an SCP monitoring interval *X* with LMC at the SCP to detect SCP overload and CG throttling at SSP Q2 to restrict the arrival rates of IN calls. Another LMC algorithm was used in the SSP to detect overload at Q2 at the end of an SSP monitoring interval *Y*. When SSP overload was detected, all incoming calls (both IN and non-IN) were then throttled equally at Q1 using a CG mechanism. The LMC overload parameters and CG interval parameter values were derived from the assumption that all IN call types had equal arrival rates at the SCP and that IN calls comprised 30% of total SSP traffic. LMC was selected as an appropriate detection routine for this investigation as it may be used in both the dynamic simulation and the steady-state mathematical model. Note, however, that while CG was used in the simulation model, PT throttles were used in the analytic model, as they are far easier to represent mathematically than CG. Twelve overload levels were defined for both the SCP and SSP detection algorithms. The operation of this strategy is shown in Figure 5.6.

The steps of the SCP congestion control scheme are shown as steps (i) to (v) in the diagram. Here, new calls entering the SSP are throttled equally by the throttle T_{SSPn}, put in place (at Q1) by the SSP at the end of its previous monitoring interval, *Y*. Then IN calls are throttled equally by the throttle T_{SCP}, put in place (at Q2) according to the overload level sent by the SCP at the end of the SCP's previous monitoring interval, *X*. All IN calls accepted at SSPs are then sent, via the SS7 to the SCP. The SCP monitors its mean load over the course of an interval *X*, and at the end of that interval, reports its overload level to all SSPs, each of which then puts the appropriate throttle in

place at Q2. The SSP control strategy works independently of this. Each SSP monitors its Q2 load over a the course of an interval *Y*, and at the end of that interval, puts throttles corresponding to its perceived overload level in place at Q1.

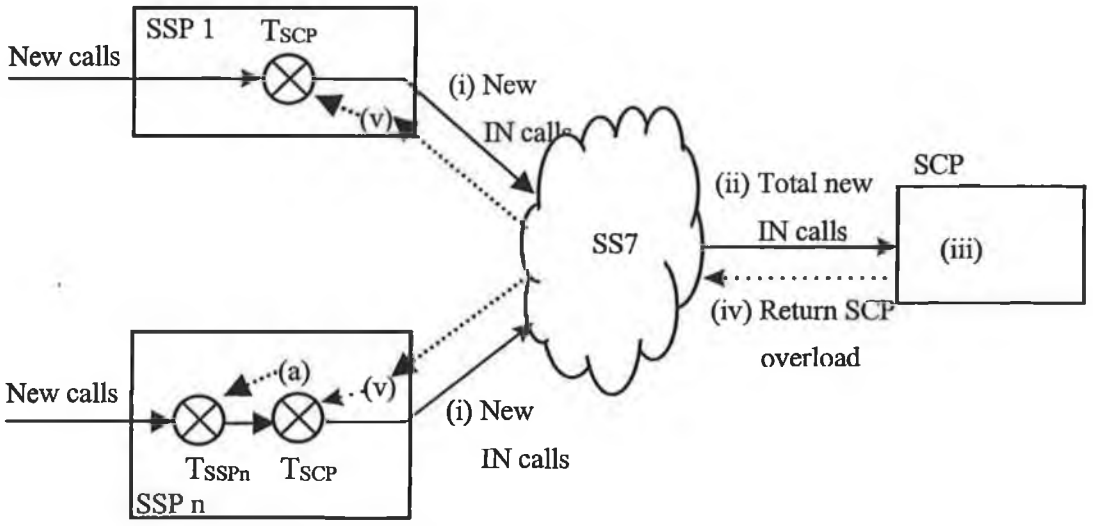


Fig. 5.6: The Independent Congestion Control Strategy

5.3.1.2 The Joint SCP/SSP Congestion Control Strategy

The joint SCP/SSP congestion control strategy (i.e. a simple example of a global IN congestion control strategy) that was developed for this investigation is made up of three separate, interacting parts - the SCP and SSP overload detection algorithms and the joint throttle. As with the independent strategy described above, LMC was used as the congestion detection method at both the SCP and at Q2 of the SSP. Also as with the independent strategy, CG throttle mechanisms are located at both Q1 and Q2. However, the operation of the throttles is quite different, in that the decision-making process as to which throttles are engaged (and the magnitude of the throttling) is more complex and is based on both SCP and SSP overload data. According to this decision-making process, there are three principle phases of operation for the CG throttle:

1. ***SCP overload.*** During this phase, SCP overload causes only IN calls to be rejected in Q2 (i.e. the number analysis stage).
2. ***First stage SSP overload.*** In this phase, SSP Q2 overload causes only the lower priority non-IN calls to be rejected in Q2 (i.e. the number analysis stage). Note that SCP throttles may or may not be in place during this phase, but non-IN and IN calls are throttled independently.
3. ***Second stage SSP overload.*** This phase is entered when selective throttling at Q2 is insufficient to alleviate the SSP overload condition (i.e. all non-IN calls are being rejected and SSP overload still exists). At this stage, minimal throttling is applied all calls equally at Q1 (during call authorisation) while the selective throttles remain in place at Q2. The aim of this is

that the SSP overload situation should be alleviated, while ensuring that the maximum possible number of calls reaches Q2, thus minimising the number of IN calls rejected.

The operation of this joint SCP/SSP congestion control strategy is shown in Figure 5.7.

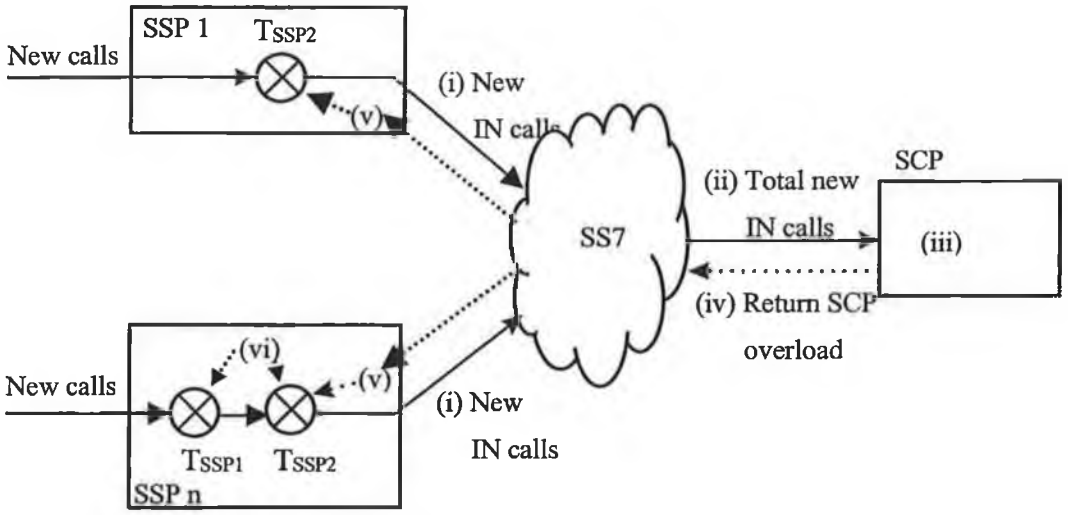


Fig. 5.7: The Joint SCP/SSP Congestion Control Strategy

Here, we see that the behaviour for SCP congestion control is identical to that defined for the independent strategy. However, for the joint (i.e. global) strategy, the SSP has the same monitoring interval as the SCP, and bases its own throttles on the overload information received from the SCP. Each SSP still puts the IN throttle in place at Q2, but it also uses this information (step (vi)) to decide, based on its own overload level, if it is sufficient to reject non-IN calls at Q2 (i.e. first stage SSP overload) or if some global throttling is required at Q1 (i.e. second stage SSP overload) in order to relieve overload at the SSP itself.

5.3.2 Results and Analysis

The criteria used to form a comparison between the independent strategy and the joint strategy are SSP load, throughput and queue length, SCP throughput, and the delays experienced by each service type. To compare these strategies, we define the requirements that both SCP and SSP loads should be maintained at 0.9 under a wide range of overload conditions. In all cases, all monitoring intervals were defined (as explained in Chapter 4, Section 4.2.1) as being of 10 seconds duration. An analysis of the operation of the two strategies will now be described under three categories, in which the operation of the joint strategy in each throttling phase will be compared with the operation of the independent strategy under the same load conditions. Note that minor discrepancies between analytical and simulation results are to be expected, as the strategies use CG in the simulation (because CG provided the best overall results, as described in Chapter 4, Section 4.3.3) and PT in the analytical model (as PT is much easier to model mathematically than CG).

5.3.2.1 SCP overload only

Here we show the results for when the offered load to the SSP is 0.85 and to the SCP is 1.2. Note that when the SSP is not overloaded, both simulation and analytical results confirm that the independent strategy and the joint strategy are equally effective. Also, the fact that there are very minor differences between simulation and analytic figures, which may be accounted for by the differences in the throttles used, verify the correctness of the two models.

Offered Load	SSP 0.85 & SCP 1.2			
Model	Analytical		Simulation	
Strategy	Indept	Joint	Indept	Joint
SSP load	0.7655	0.7655	0.765	0.765
SSP throughput	0.7525	0.7525	0.753	0.755
SCP load	0.9	0.9	0.9	0.91
SSP queue length	3.26	3.26	3.27	3.27
Non-IN delay	0.237	0.237	0.24	0.24
Freephone delay	0.617	0.617	0.635	0.65
Televoting delay	1.48	1.48	1.6	1.725

Table 5.2: SCP overload, No SSP overload

5.3.2.2 SSP overload

Results are described for two load situations for which the SSP is overloaded:

- The offered load to the SSP is 1.2, of which 0.325 consists of IN calls, offering a load of 0.52 to the SCP, i.e. the SSP is overloaded and the SCP is not (i.e. overload is caused by non-IN calls). The load levels here are sufficient to cause the joint strategy to enter first stage SSP overload.
- The offered load to the SSP is 1.2 Erlangs, of which 1.05 comprises IN calls (offering a load of 0.78 to the SCP). In this case, overload is caused by IN calls. Here, the joint strategy enters second stage SSP overload, and will cause minimal throttling of all calls at Q1.

In the first case, the SCP will never become overloaded and will accept all calls offered to it. The SSP, however, is overloaded and will either selectively throttle non-IN calls, for the joint strategy, or impartially throttle all calls, for the independent strategy. For this scenario, the throughput of the SSP is lower by 0.02 for the joint strategy than for the independent strategy - this is due to the capacity which is required to progress non-IN calls to the point where they may be identified as

such, before being rejected. This processing overhead for the joint strategy is to be expected, due to its added complexity, but Figure 5.8 shows the advantage of using the joint control scheme - note that SCP load is 0.12 greater for the joint strategy than for the independent strategy. This is a significant gain in a system where IN calls are prioritised. The analytical model validates these results in Table 5.3.

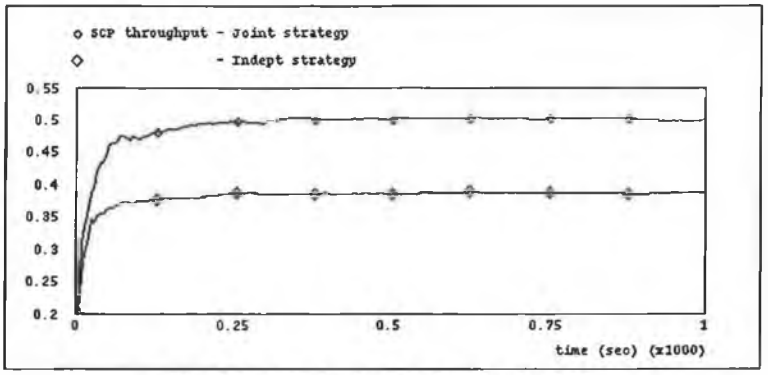


Fig. 5.8: SSP load = 1.2, SCP load = 0.52

Offered Load	SSP 1.2 & SCP 0.52			
Model	Analytical		Simulation	
Strategy	Indept	Joint	Indept	Joint
SSP load	0.93	0.93	0.93	0.93
SSP throughput	0.91	0.89	0.907	0.889
SCP load	0.4	0.52	0.38	0.5

Table 5.3: SSP overload, no SCP overload

For the second case, when overload of the SSP is caused by IN calls, the advantage of using the joint strategy to maximise the number of IN calls accepted is decreased, as the joint strategy must reject almost as many IN calls as the independent strategy does, in order to relieve the overload situation. The result is that the SCP load for the joint strategy is now only 0.06 Erlangs greater than that for the independent strategy. On the other hand, the fact that the joint strategy must start to reject all calls equally at SSP Q1 reduces significantly the processing overhead generally associated with this strategy. In fact, Figure 5.9 shows that the disparity between throughput values for the two strategies is almost eliminated.

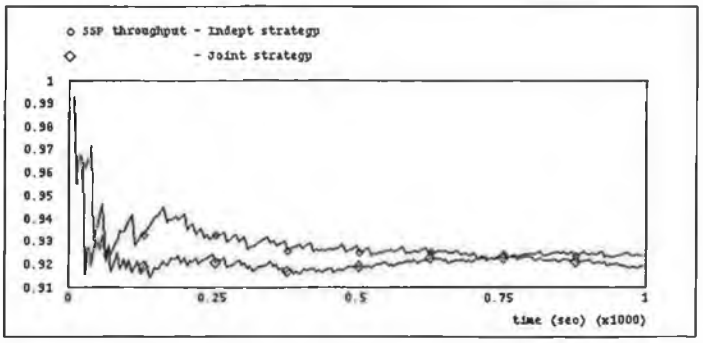


Fig. 5.9: SSP load = 1.2, with IN calls comprising 1.05 Erlangs

A comparison of simulation and analytical results of this scenario is provided in Table 5.4 below.

Offered Load	SSP 1.2 & SCP 0.78			
Model	Analytical		Simulation	
Strategy	Indept	Joint	Indept	Joint
SSP load	0.93	0.93	0.93	0.93
SSP throughput	0.918	0.913	0.925	0.92
SCP load	0.696	0.78	0.704	0.766

Table 5.4: SSP load = 1.2, with IN load 1.05

In other words, when the SSP alone is overloaded, there is an automatic tradeoff within the joint strategy, such that noticeable increase in SSP processing requirements results in considerable increase in IN call acceptances, whereas if the state of the SSP overload is such that a considerable increase in IN call acceptances is not possible, the SSP processing requirements are reduced automatically.

5.3.2.3 Both SSP and SCP overloaded

For this scenario, the offered load to the SSP is 1.2 Erlangs, of which 0.352 consists of IN calls (offering a load of 1.3 to the SCP), i.e. both SCP and SSP are overloaded. The results for this case are presented in Table 5.5.Note that the difference in SCP loads achieved by the two strategies is smaller (0.02), but this is also reflected in the difference in SSP throughputs, which is now only approximately 0.005.

The operation of the two strategies is quite interesting for this scenario. The joint strategy, as usual, has an SSP processing overhead of approximately 0.02, but due to the fact that the throttling it puts in place is based on the state of both SSP and SCP, it succeeds in keeping both the SSP and SCP load levels close to the defined threshold. For the independent strategy, however, the SCP

causes suitable throttles to be put in place at SSP Q2, while the SSP (not knowing the state of the SCP) puts unnecessarily stringent throttles in place at Q1, resulting in the unnecessary and undesired rejection of IN calls (and lower SSP load and throughput characteristics). Therefore, the independent strategy overprotects both the SCP and the SSP.

It is also interesting to note that, in the simulation model, the independent strategy exhibits significantly greater oscillations in SSP load and throughput than the joint strategy. This is due to the fact that, at any given time, the SSP does not know the current state of the SCP, and therefore any significant variations in SCP load levels result in oscillations in SSP throughput over the course of a number of proceeding monitoring intervals.

Offered Load	SSP 1.2 & SCP 1.3			
Model	Analytical		Simulation	
Strategy	Indept	Joint	Indept	Joint
SSP load	0.91	0.92	0.91	0.92
SSP throughput	0.902	0.897	0.905	0.9
SCP load	0.88	0.9	0.885	0.905

Table 5.5: SSP and SCP overload

In summary, at SSP load levels below 1.0, results for both the joint SSP/SCP congestion control strategy and the independent strategy show them to be equivalent, irrespective of SCP load levels. When only the SSP is overloaded, the comparative behaviour of the two strategies depends on the input traffic mix. However, the joint strategy consistently accepts significantly more IN calls than the independent strategy, at a cost of not more than 0.02 SSP Erlangs (due to the requirement to process calls through to number analysis prior to rejection).

When both SSP and SCP are overloaded, again there is an overhead of not more than 0.02 Erlangs at the SSP for the joint strategy, but in return, the number of IN calls processed at the SCP is maintained at the threshold, independent of the SCP overload level, while the independent strategy consistently maintains SCP load significantly below the threshold (i.e. it overprotects the SCP). The joint strategy also has the added advantage that, as it is simultaneously aware of the state of both the SCP and SSP, throttles are put in place which protect both physical elements at all times, while for the independent strategy, the SSP algorithm is only aware of the state of the SCP defined by the throttles which were put in place at the end of the preceding interval. The result of this is that variations in traffic load or mix may cause oscillations in SSP load and throughput for the independent strategy, but do not affect the efficiency of the joint strategy.

5.3.3 Conclusions

Here we investigated the operation of a joint SCP/SSP congestion control strategy with an independent strategy. The results (published in [Lodge96]) showed that when independent congestion control strategies are used to protect the different elements of an IN, they successfully meet the basic requirements placed on such strategies – i.e. they protect their elements with small processor overheads. However, they lack flexibility, in that the interworking between the different control algorithms results in inefficient overall performance. This was proven by comparing classic congestion control algorithms, operating independently at the SSP and SCP, with a very simple joint algorithm that selectively throttled input traffic based on the state of both elements. The results demonstrated that even a simple global IN congestion control strategy is, for all traffic mixes and loads, either equivalent to or superior to a strategy in which the SCP and SSP are protected independently.

Independent strategies may be both flexible and fair, but again, only within the element being protected. Any global concepts of flexibility or fairness cannot, in general, be supported by these kinds of strategies and concepts such as maximum network resource utilisation and prioritisation are not naturally addressed by this class of strategy. For example, in this section, it was very simple to define the joint strategy so that it encompassed a simple priority system, where all IN calls had higher priority than all non-IN call types. The results proved that, at all load levels and for all traffic mixes, this priority system was adhered to, at the expense of a small processing overhead at the SSP. Adapting independent strategies to provide the same level of support for prioritisation would be non-trivial and would remain subject to problems when required to interwork with each other (as will always be the case in IN).

The principle conclusion to be drawn from the work described in this section is that there is a need for a network-wide congestion control strategy in the IN which provides controls for both SCPs and SSPs by throttling both IN and non-IN call types appropriately. Only through use of such a strategy is it possible to maximise the performance of the entire network. A global IN congestion control strategy would also be easily extensible to encompass all desired aspects of a congestion control strategy, including scalability, fairness and flexibility. Note however, that the joint strategy used in this investigation is not proposed as a solution for global IN performance management, as it is neither scalable (CCC and CG both require parameters to be defined which are dependent on the capacity of the resource being monitored and are therefore innately unscalable) nor elegant. It was designed merely as a tool to verify the necessity for a global IN strategy and to motivate the specification of a comprehensive solution.

Section 5.4 presents an elegant strategy for global IN performance management which meets all basic and desired aspects of any congestion control strategy, and verifies the usefulness of this solution in Section 5.5 through the use of the IN analytical model. Chapter 6 will compare this solution with existing strategies (using the simulation model) in order to prove its superiority.

5.4 The Optimisation-based Global IN Congestion Control Strategy

The congestion control strategy presented here consists of an algorithm which, at defined intervals, uses an optimisation program to find the best possible percent thinning throttling arguments for each type of input call in order to maximise the revenue during the next interval. However, this optimisation program must also satisfy a number of constraints. These include:

- load constraints on the SSP and SCP (a form of load measure control), and
- constraints ensuring that the weights or priorities of the different call types are reflected in the defined throttles.

In this manner, revenue will be maximised over a time interval, while traffic will be throttled in such a way that the throttling levels for each type of input call will preserve the *weights* defined for the calls while ensuring that load thresholds at the SSP and SCP are not exceeded. Note that this strategy may be generalised easily to address congestion issues for any system with multiple input traffic types - it is only the addition of the SCP constraint that causes it to address IN congestion specifically.

5.4.1 Defining the Mathematical Terms to be used in the Strategy Specification

Before the devised optimisation-based congestion control strategy is described, it is necessary to make a few mathematical assumptions, in order to clearly define and scope the terms which will be used in the formulae associated with the strategy – i.e. the formulae will be expressed in the terms of the analytic IN model. This in no way detracts from the generality of the approach – it merely aids understanding in the specification of the strategy.

The following assumptions were made:

- There is one SCP, one SDP and N SSPs in the system,
- All service times are exponentially distributed,
- No calls are rejected at Q1, i.e. $P_n^a = 1.0$,

resulting in the specialisation of the general decomposition equations defined in Section 5.2.3.2, as shown below. Note that again, all parameters on the right hand side are as defined for time interval $[t\text{-}T,t]$, with the exception of P_n^a and $p_{n,i}^a$, which are as defined at time $(t\text{-}T)$.

For $Q1_n$ & $Q2_n$: $Ka_{Q1_n}(t) = C_{Q1_n}(t) = Ka_{Q2_n}(t) = 1$

$$C_{Q2_n}(t) = 1 - 2\rho_{Q2_n}{}^2 + 2\lambda_n \sum_i \lambda_{n,i} \left(\frac{p_{n,i}^a}{\mu^2_{acc_{n,i}}} + \frac{(1-p_{n,i}^a)}{\mu^2_{rej_n}} \right)$$

For $Q3_n$:

$$Ka_{Q3_n}(t) = \frac{1}{\lambda_{Q3_n}} \left[\left[(C_{Q2_n} - 1)\left(\sum_{k=1,4} \frac{p_{n,k}^a \lambda_{n,k}}{\lambda_n} \right) + 1 \right] \left(\sum_{k=1,4} p_{n,k}^a \lambda_{n,k} \right) + \left[(C_{SCP} - 1)\left(\sum_{j=2,3,5} \frac{p_{n,j}^a \lambda_{n,j}}{\lambda_{SCP}} \right) + 1 \right] \left(\sum_{j=2,3,5} p_{n,j}^a \lambda_{n,j} \right) \right]$$

$$C_{Q3_n}(t) = \rho_{Q3_n} + Ka_{Q3_n}(t)(1 - \rho_{Q3_n})$$

For the SCP:

$$Ka_{SCP}(t) = \frac{1}{\lambda_{SCP}} \left[\sum_{n=1}^{N} \left[(C_{Q2_n}(t) - 1)\left(\frac{1}{\lambda_n} \sum_{j=2,3,5} p_{n,j}^a \lambda_{n,j} \right) + 1 \right] \left(\sum_{j=2,3,5} p_{n,j}^a \lambda_{n,j} \right) + C_{SDP}(t)\lambda_{SDP} + \sum_{n=1}^{N} C_{IPQ_n}(t)\lambda_{IPQ_n} \right]$$

$$C_{SCP}(t) = \rho_{SCP} + Ka_{SCP}(t)(1 - \rho_{SCP})$$

For the SDP:

$$Ka_{SDP}(t) = 1 + (C_{SCP}(t) - 1)\left(\frac{1}{\lambda_{SCP}} \sum_{n=1}^{N} \sum_{j=2,3,5} p_{n,j}^a \lambda_{n,j} \right)$$

$$C_{SDP}(t) = \rho_{SDP} + Ka_{SDP}(t)(1 - \rho_{SDP})$$

For IPQ_n:

$$Ka_{IPQ_n}(t) = 1 + (C_{SCP}(t) - 1)\left(\frac{2p_{n,3}^a \lambda_{n,3}}{\lambda_{SCP}} \right)$$

$$C_{IPQ_n}(t) = \rho_{IPQ_n} + Ka_{IPQ_n}(t)(1 - \rho_{IPQ_n})$$

5.4.2 Capturing the Requirements on the Global IN Congestion Control Strategy

Before describing the optimisation-based global IN congestion control strategy, let us first summarise the requirements which were specified on it, as described in Section 5.1. Obviously, the basic requirements are that an overload control strategy must be able to protect IN resources under any load conditions and must use processing resources in an efficient manner. Highly desirable characteristics of the strategy also include:

- Scalability: an algorithm should not need to be substantially reconfigured if the size of its resource changes, e.g. if the same algorithm should reside at all SSPs, it should not require much "tweaking" to target it to each SSP. For example, while requiring that the values for a few parameters should be set is acceptable, the need to run exhaustive simulations to establish, for each SSP, the number of arriving calls which constitute overload (for CCC) or the gap

intervals which should be put in place in a CG throttle for a given overload level is highly undesirable.

- Flexibility: a well-designed algorithm should be easily customised to include factors such as different call priorities, different requirements on fairness or other non-functional requirements.

- Fairness: If overload is caused by an excess of calls of one particular service type, only calls of that type should be rejected (i.e. Focussed Destination Overload Control - FDOC). Also, all subscribers to a particular service type should have an equal chance of acceptance – i.e. all sources should be throttled proportionally to their size and arrival rates.

To ensure that the solution proposed addresses all the above requirements, a number of factors were specified which must be taken into account when defining the algorithm. In the area of flexibility, these sample factors are defined by the requirements of users and operators of the network. For the operator, it is desirable to maximise revenue while maintaining IN integrity. IN customers expect the network operator to provide a pre-agreed Quality of Service (QoS) level, while all users require that call setup takes place as quickly as possible. Other factors may also be equally relevant, but due to the constantly increasing number of new types of service on offer in telecoms networks, it would be very difficult to provide an exhaustive list of them. Therefore, in designing our strategy, we have chosen to base our priority-driven system on the following factors:

- revenue per call,
- load requirements per call, and
- IN QoS agreements.

The aim of this chapter is to find a strategy that balances these factors and provides a good compromise in order to satisfy all users. Note that seeking a good balance between these factors curtails, to a certain extent, the capability to strictly enforce either service or subscriber fairness. Instead, it is required that the designed strategy maintains fairness *within the bounds* of the balance between the above factors. For example, if a service type which has high revenue per call, low load requirements and which is based on strict QoS agreements causes an overload, calls of this type should logically still be given priority over less important calls. However, accepting this, the system should never reach the state where *all* calls of a particular type are rejected due to an overload of calls of a higher priority – this would be highly unfair. Maintaining a balance between call types should ensure that this situation never arises.

5.4.3 Introducing the Concept of Call Weights

Maintaining a balance between call types is accomplished through the definition of call weights. When using simple priority-based schemes, we lose the information as to the relative importance of calls of different priorities. We therefore introduce *call weights* as a factor of the relative importance of different call types (this is, in fact, a generalisation of priorities). For this work, we choose to focus on the call setup revenue, call processing requirements, and QoS agreements as the factors that contribute to a call type's weight. In order to place numerical values on QoS agreements, we assign numerical QoS *levels* which may be set by the service provider to capture the relative strictness of the requirements of different service users - these levels may be assigned based on practical data (such as acceptable delays) or on more abstract non-functional requirements (e.g. the importance of a customer). This is at the discretion of the service provider.

The weight of each call type should be a function with the following properties:

- For all other factors constant, if the value of revenue is increased, the weight should increase (but not necessarily proportionally),
- For all other factors constant, if the value of QoS is increased, the weight should increase,
- For all other factors constant, if call processing requirements are increased (captured using the value of service time), the weight should decrease,

For this work, the relative weight for call type *i* was defined as ω_i, where

$$\omega_i = \frac{R_i q_i \mu_i}{\sum_k R_k q_k \mu_k}$$

i.e. we assign weights to each of the traffic types by finding the normalised product of their setup revenue, R_i, their total (i.e. at both SSP and SCP) load requirements (here expressed as the inverse of service time, service rate μ_i) and their pre-agreed QoS level q_i.

For the five call types that are implemented in the model, a number of assumptions are made:

- International call types have highest revenue, with televoting second and freephone and local with the lowest revenue,
- IN calls require approximately 2.5 times the processing capacity of non-IN calls at the SSP,
- All IN calls receive the same amount of processing capacity each time they enter the SCP,
- International freephone has the highest QoS level, with televoting second, freephone third, and non-IN calls having the lowest priority as no QoS levels are defined for them.

Examples of possible normalised weight assignments are shown in Table 5.6, as are the ratios of these weights.

Service	Call type	ω_i	$\frac{\omega_i}{\omega_5}$
International	1	0.489	12.225
International freephone	2	0.313	7.825
Televoting	3	0.1	2.5
Local	4	0.058	1.45
Freephone	5	0.04	1.0

Table 5.6: Assignment of Weights

5.4.4 Specification of the Optimisation-based IN Congestion Control Strategy

As stated in Section 5.4 and described in [Lodge97], the proposed global IN congestion control strategy takes the form of two optimisation algorithms, one of which is located in the SCP and the other in all SSPs. These algorithms are very similar, but are specialised to the resource in which they reside. They interoperate in such a manner as to provide premium IN performance, while ensuring that no network elements become overloaded. The sequence of operation and interaction of these algorithms is shown in Figure 5.10 and will now be explained.

During an interval of length T, each SSP n has a mean arrival rate of $\lambda_{n,i}$ for each service type i (i =1..5). These arrivals are subject to any throttles put in place at $Q2_n$ at the end of the previous interval (step (i) from the diagram) and all accepted IN calls are sent to the SCP via the SS7 network. We define $\lambda^{new}_{SCP,j}(t)$ as the mean arrival rate during the period [t-T, t] of new calls of type j (where $j \in J$ where $J = \{2,3,5\}$ for IN calls) as calculated at the SCP (step (ii)) at time t. This may be estimated as $\lambda^{new}_{SCP,j}(t) \approx \sum_n p^a_{n,j}(t-T)\lambda_{n,j}$.

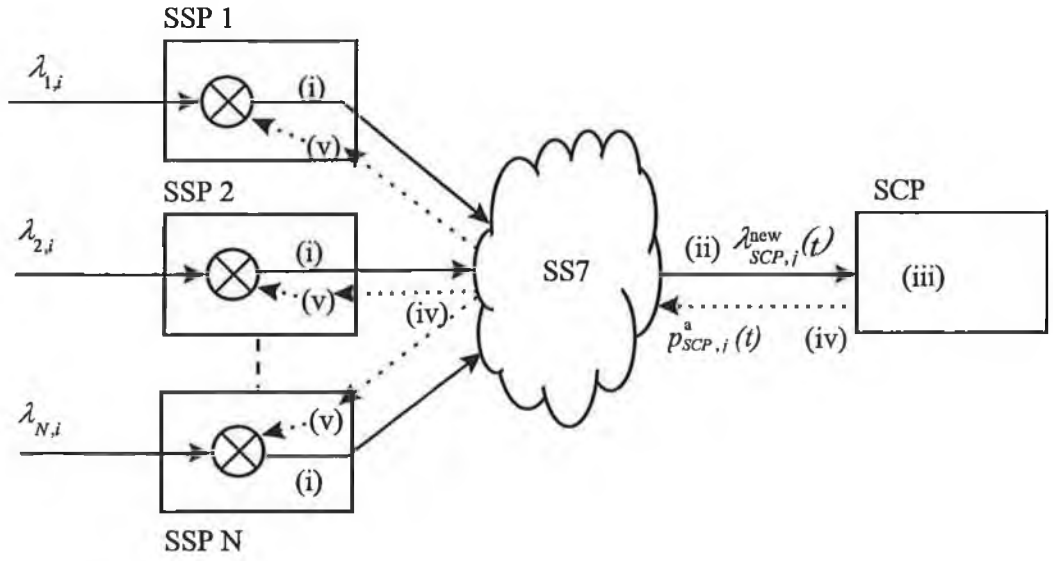


Fig. 5.10: Overview of the New Global IN Congestion Control Strategy

Optimisation at the SCP (step (iii)). The SCP algorithm, at time *t*, attempts to set values on the SSP PT throttles in order to maximise revenue for IN calls in the next interval, while meeting SCP load constraints and honouring the weights of different traffic types. To do this successfully, the optimisation must be based on the total arrivals of IN calls to the network during the previous *T* seconds. Note, however, that the SCP has no information regarding the actual number of arrivals to the system (i.e. to the SSPs) and must therefore estimate this value based on the information that is available to it, namely $\lambda^{new}_{SCP,j}(t)$ and the values of the throttles it defined at the end of the previous interval $p^{a}_{SCP,j}(t-T)$. This estimated total for each IN call type at time *t* may be evaluated using the formula:

$$\lambda^{est}_{total,j}(t) \approx \frac{\lambda^{new}_{SCP,j}(t)}{p^{a}_{SCP,j}(t-T)}$$

The optimisation program to maximise the revenue over IN call types may be expressed as follows for *J* IN call types, each with known associated revenue per call R_j, a probability of acceptance $p^{a}_{SCP,j}(t)$ (the throttles to be determined for the next control interval) and the estimated new call arrival rate $\lambda^{est}_{total,j}(t)$, used because, to maximise revenue, all calls should be counted once only, as they may be charged for only once.

$$\underset{p^{a}_{SCP,1}(t)\ldots p^{a}_{SCP,J}(t)}{\text{Maximise}} \quad \sum_{j\in J} R_j p^{a}_{SCP,j}(t)\lambda^{est}_{total,j}(t)$$

The constraints on this revenue maximisation are:

(i) SCP Load constraint: $\frac{1}{\mu_{SCP}}\sum_{j} p^{a}_{SCP,j}(t)e_{SCP,j}\lambda^{est}_{total,j}(t) \le Thr_{SCP}$,

(ii) Bounds on $p^{a}_{SCP,j}(t)$: $0 \le p^{a}_{SCP,j}(t) \le 1, \ \forall j$,

(iii) Weight constraints: $1 \le \frac{p^{a}_{SCP,2}(t)}{p^{a}_{SCP,j}(t)} \le \frac{\omega_2}{\omega_j}, \ j=3,5$, where $\omega_j = \frac{R_j q_j e_{SCP,j}\mu_{SCP}}{\sum_{k=2,3,5} R_k q_k e_{SCP,k}\mu_{SCP}}$,

where μ_{SCP} is the processing rate at the SCP, Thr_{SCP} is the overload threshold for the SCP and all other parameters are as previously defined. In Constraint (iii), the ratio of weights is defined in terms of call type 2. This is because this call type has the highest priority of the IN call types. The optimisation algorithm will execute every *T* seconds and will, at time *t*, based on arrival rates in the interval [*t-T,t*], calculate the optimal (in terms of IN revenue) probabilities of acceptance $p^{a}_{SCP,j}(t)$ for the interval [*t,t+T*]. These *J* arguments (corresponding to call types 2, 3 and 5 in our model) are returned to the SSPs in the network (step (iv)) and are used to define the *upper bounds* on the percentage of calls accepted for each IN call type. These arguments may not be used

explicitly as the PT arguments as they do not take into account the state of each SSP and therefore may detrimentally affect the performance of these elements.

Optimisation at the SSP (step (v)). The optimisation algorithm at the SSPs functions in a manner similar to that described for the SCP. Here, the acceptance probabilities $p^{a}_{SCP,j}(t)$ defined by the SCP optimisation algorithm are used. Again, the revenue is maximised (note that, at the SSP, non-IN calls must also be taken into account) subject to load and weight constraints. The objective function may be stated as follows, where $I = 5$:

$$\underset{p^{a}_{n,1}(t)\ldots p^{a}_{n,I}(t)}{\text{Maximise}} \quad \sum_{i=1}^{I} R_i p^{n}_{a,i}(t)\lambda_{n,i}(t)$$

with the constraints:

(i) SSP_n Load constraint : $\sum_{i=1}^{I} \lambda_{n,i}(t)\left(\frac{p^{a}_{n,i}(t)}{\mu_{acc_n,i}} + \frac{(1-p^{a}_{n,i}(t))}{\mu_{rej_n}}\right) \le Thr_{SSP_n}$,

(ii) Bounds on $p^{a}_{n,i}(t)$ for non - IN calls : $0 \le p^{a}_{n,i}(t) \le 1$, where i=1,4 ,

(iii) Bound on $p^{a}_{n,i}(t)$ for IN calls : $0 \le p^{a}_{n,i}(t) \le p^{a}_{SCP,i}(t)$, where i=2,3,5 ,

(iv) Weight constraints : $1 \le \frac{p^{a}_{n,1}(t)}{p^{a}_{n,i}(t)} \le \frac{\omega_1}{\omega_i}$, $i = 2,3,4,5$, where $\omega_i = \frac{R_i q_i \mu_{acc_n,i}}{\sum_{k=1}^{5} R_k q_k \mu_{acc_n,k}}$,

where $\mu_{acc_n,i}$ is the processing rate at SSP n for accepted call type i, Thr_{SSP_n} is the user-defined overload threshold for SSP n, and all other parameters are as previously defined. Note that Constraints (i) and (iv) protect the SSP from overload while preserving priorities between call types and that Constraint (iii) ensures that SCP load requirements are not exceeded.

Note that the objective functions and all constraints at both the SCP and SSPs are linear - therefore, these are Linear Programming Problems, as described in Chapter 3. Other constraints could also be included, as the optimisation algorithm is very flexible, but it is recommended that they should be included only if linear, because the inclusion of any non-linear constraints would render the optimisation task much more complex and it would thus require much more processing power. For this reason, a constraint on service delays, which would be non-linear, was not included. This should, however, not be a problem, as the other constraints should ensure that the network does not become overloaded and that therefore the service delays should be within acceptable bounds. Additionally, load constraints at the SCP and SSP were based explicitly on load levels for computational ease only - they could easily be amended so that the overload thresholds are defined by the number of call arrivals (call count control) or the mean queue length (a form of queue length control). Note also that, as with all strategies that estimate congestion

levels over a fixed interval, there will always be a delay of a maximum of T seconds before the algorithm responds to the onset of congestion. Further, as no requests may be rejected until they reach Q2, it is possible that Q1 may become overloaded independently. Therefore, it is necessary to place a simple throttle (e.g. a rate control mechanism) at Q1 to reject just enough traffic to protect this queue.

5.5 Operation of the Global IN Congestion Control Strategy

To show the efficient operation of the control strategy, we must present the results acquired from the analytic model in a number of different input load scenarios. In all cases, the interval T is defined as 10 seconds, while $Thr_{SSP} = Thr_{SCP} = 0.8$.

5.5.1 Scenario 1: Stationary Case.

As a first step, to prove the validity of the analytic work, we state that, in the stationary case, i.e. when input traffic levels are constant for all call types and at levels sufficient to cause overload, processor loads, revenue and service delays all converge to their optimal levels within two iterations/intervals. Controls are put in place at the end of the first interval to deal with the congestion situation, and are amended slightly at the end of the second interval to cater for retrials. This verifies that the optimisation algorithm deals with changes in overload levels quickly and efficiently.

5.5.2 Scenario 2: SSP Overload due to One Call Type

Here, SSP overload was induced by increasing the arrival rate of international freephone calls linearly over successive intervals for 60 iterations and then decreasing it sharply, while maintaining constant arrival rates for all other call types (shown in Figure 5.11). Note that the SCP service rate was defined so that the SCP would not become overloaded and adversely affect the results for this scenario.

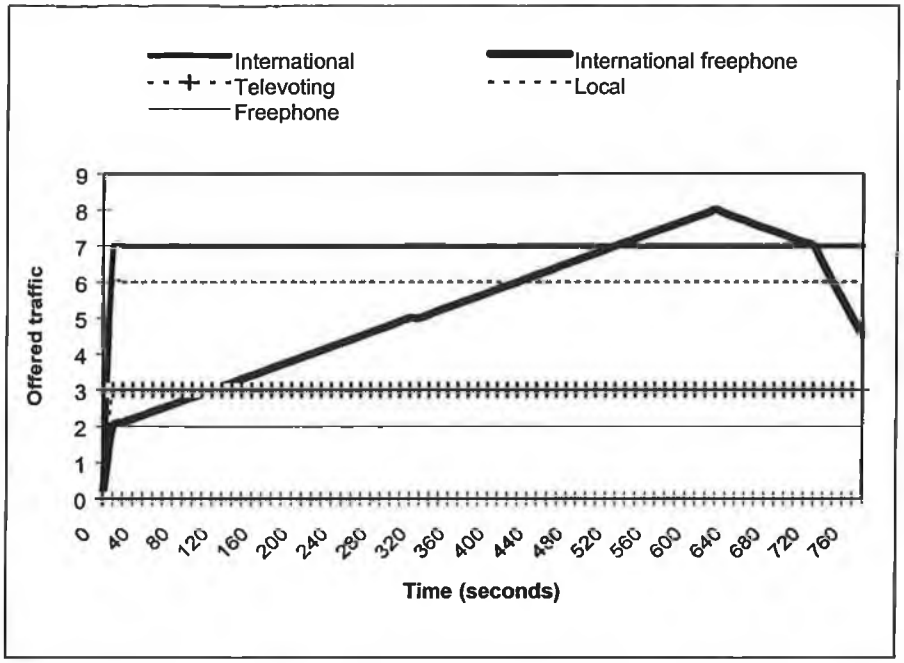


Fig. 5.11: Offered Traffic Causing SSP Overload

Constraint (iv) of the SSP optimisation algorithm forces rejection of call requests of lower weight, while also maintaining the ratios between probabilities of acceptance of these calls, as demonstrated in Figure 5.12. Note that, Constraint (i) forces televoting and freephone calls to be restricted prior to local calls – this is because of the greater SSP processor requirements associated with IN calls. When Constraint (iv) becomes active between freephone and televoting, local calls start being restricted. This continues until Constraint (iv) again becomes active for local calls, and then a minor quantity of international freephone calls are rejected.

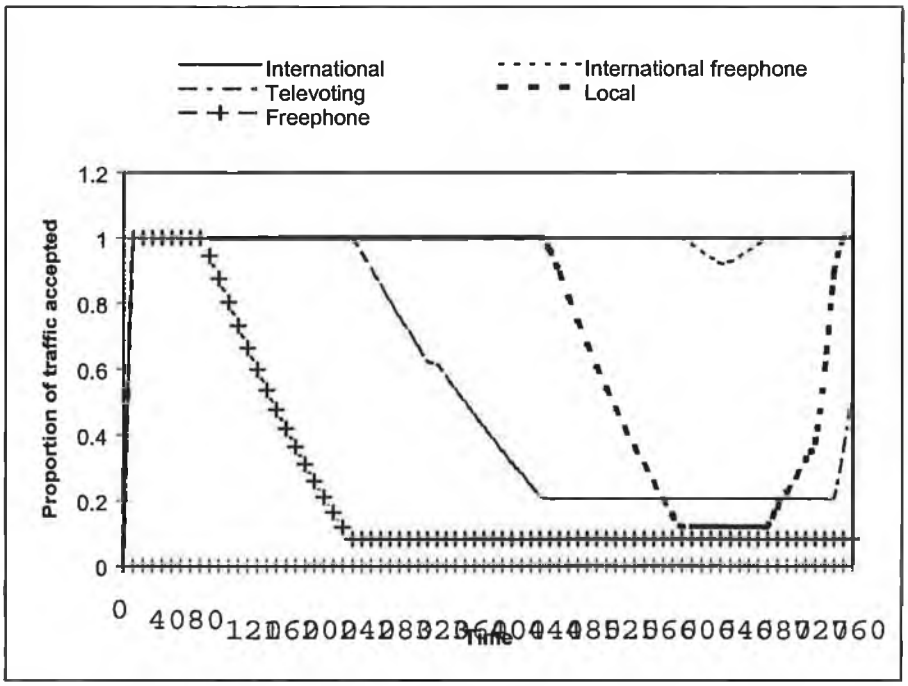


Fig. 5.12: Proportion of Traffic Accepted under SSP Overload

Constraint (i) forces processor load for the SSP stay at the threshold - see Figure 5.13. Note that at time t=440, SCP load reaches its minimum value - this is due to the rejection of freephone and televoting calls. However, at this point, Constraint (iv) becomes active and rejections of these calls stabilises and local calls are rejected, while international freephone arrivals continue to rise, thus prompting the rise in SCP load after this point. SCP load does not begin to drop until after the arrival rates for international freephone fall, and its rate of decent is slowed by the gradual increase in televoting calls accepted. Figure 5.14 shows that the revenue follows the variation of international freephone in a non-linear manner, as would be expected.

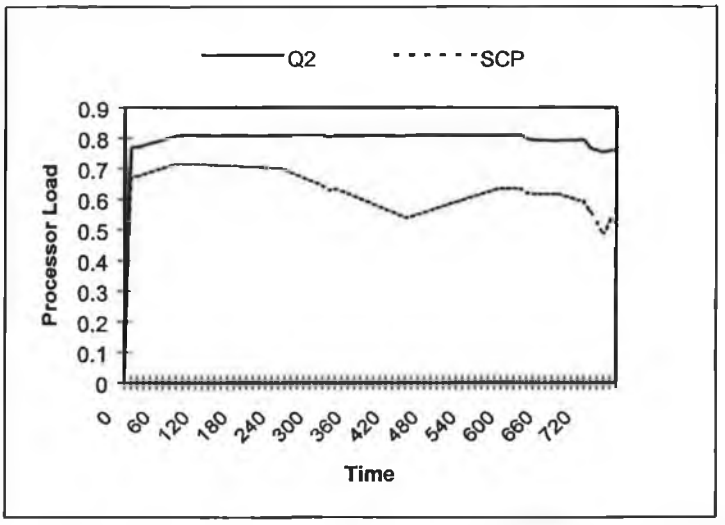


Fig. 5.13: SCP and SSP Processor Loads during SSP Overload

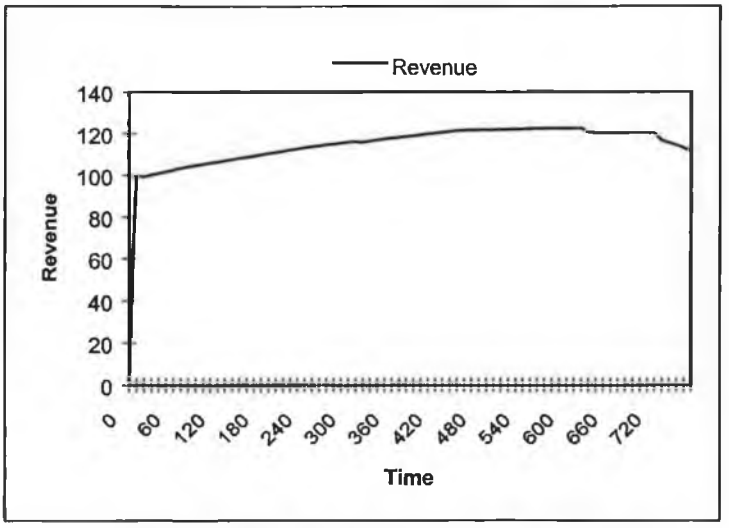


Fig. 5.14: Revenue during SSP Overload

5.5.3 Scenario 3: SCP Overload due to One Call Type

In this case, traffic arrival levels were kept sufficiently low to ensure that the SCP became overloaded prior to the SSP. Televoting arrival rates increased linearly for 66 iterations before

dropping off sharply. The results are very interesting. SCP Constraint (i) ensures that the SCP load threshold is not exceeded, while due to the fact that televoting arrivals far exceeded freephone arrivals, Constraint (iii) forces only the rejection of televoting calls - i.e. the calls that caused the overload. The variation in televoting acceptances follows the variations in the arrival rates for this call - note how the acceptances decrease non-linearly for linear increases in arrival rates, and that, at the point where televoting arrival rates begin to decrease (i.e. at t=660), televoting acceptances begin to ascend at the same rate in a non-linear fashion. These results are shown in Figures 5.15.

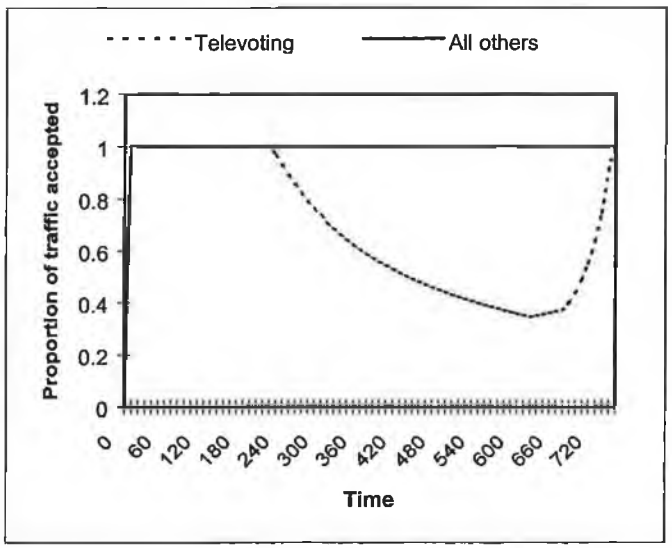


Fig. 5.15: Proportion of Traffic Accepted at SSP during SCP Overload

Figure 5.16 shows the processor loads of the SCP and Q2 over the same period - note that, due to the fact that the SCP exhibits overload symptoms before the SSP does, protection of the SCP prevents the onset of congestion at the SSP.

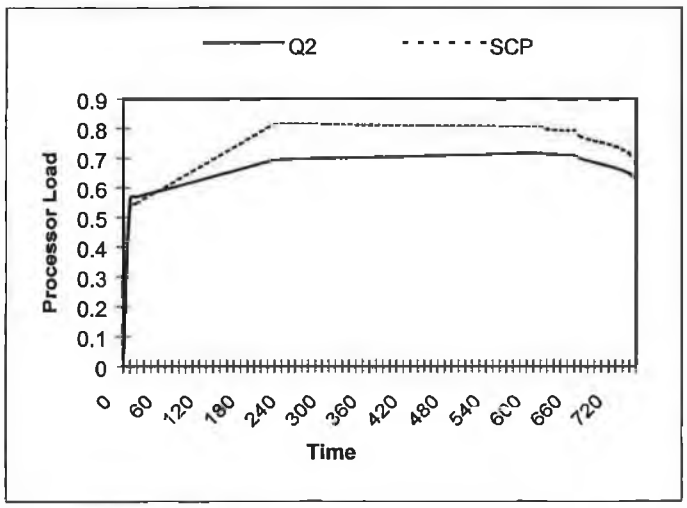


Fig. 5.16: SSP and SCP Processor Loads during SCP Overload

5.5.4 Scenario 4: General Overload

In this scenario, the arrival rate for all calls increased non-linearly (in steps). The result was the rejection of all call types with the ratio of probabilities of arrival between call types being maintained by SCP Constraint (iii) and SSP Constraint (iv). This is shown in Figure 5.17 - note how SSP Constraints (i) and (iv) again force the rejection of televoting prior to the rejection of local calls and also that no local calls are rejected until SSP Constraint (iv) becomes active for freephone and televoting.

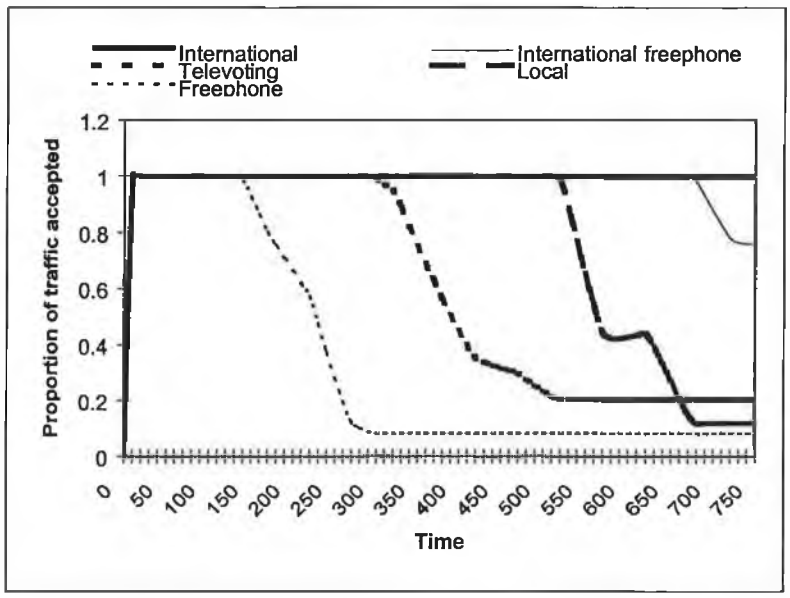


Fig. 5.17: Proportion of Traffic Accepted at SSP during General Overload

Processor load at the SSP is maintained at the threshold by SSP Constraint (i) while processor load at the SCP changes with the variation in acceptance levels for the different call types, but consistently remains below its threshold (see Figure 5.18).

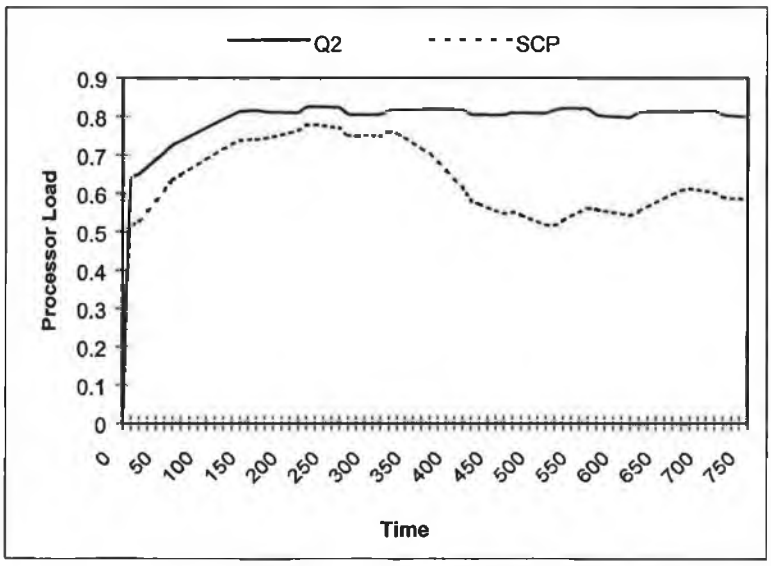


Fig. 5.18: SSP and SCP Processor Loads during General Overload

Changes in the revenue follow variations in the non-linear arrival rates, but with smoother variations, as shown in Figure 5.19.

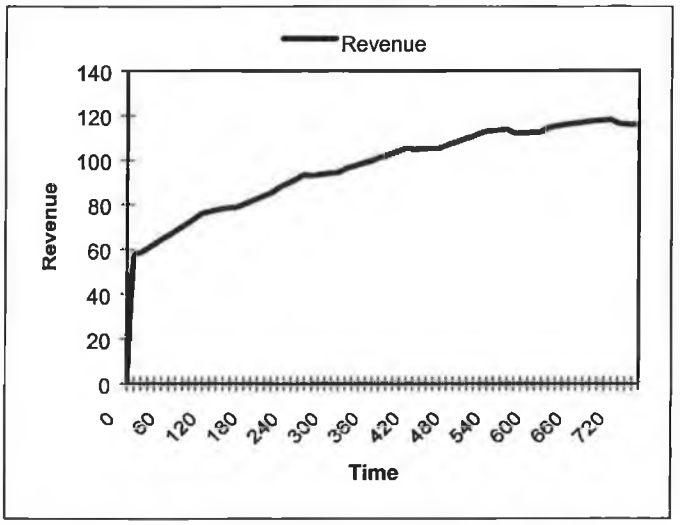


Fig. 5.19: Revenue during General Overload

Figure 5.20 demonstrates the service delays experienced by each traffic type - note that the delays are greater for large SCP load (shown in Figure 5.18) and then decrease as SCP load decreases due to the throttling of televoting and international freephone traffic. However, at all times, the existing constraints ensure that the delays for each service type are within acceptable limits as defined by [MacDonald94]. This confirms that the explicit inclusion of a constraint on service delays in the optimisation algorithm is unnecessary.

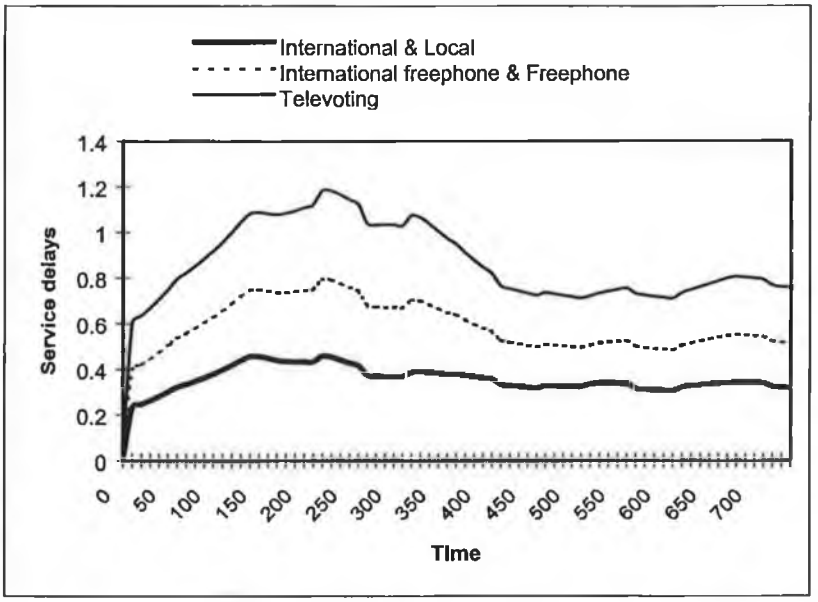


Fig. 5.20: Service Delays during General Overload

5.5.5 Scenario 5: Overload due to Bursty Televoting Traffic

To examine the operation of the control strategy when subjected to short bursts of high traffic loads, an input was provided which generally did not cause any load constraints to be exceeded. However, at irregular intervals, the arrival rate of televoting calls was increased by a factor of 4 for 60 seconds. All other traffic arrival rate remained constant for the duration of the simulation. The load results, shown in Figure 5.21 show that, at the end of the interval following the onset of the burst, the system responds by throttling televoting calls (due to SCP Constraint (iii)) to reduce the loads on both SSP and SCP to approximately threshold levels. When the burst traffic ceases, the system again responds at the end of the succeeding interval by eliminating the televoting throttle, thus restoring the original loads. Note, in Figure 5.22, that the resultant revenue closely follows the load curve of the SSP.

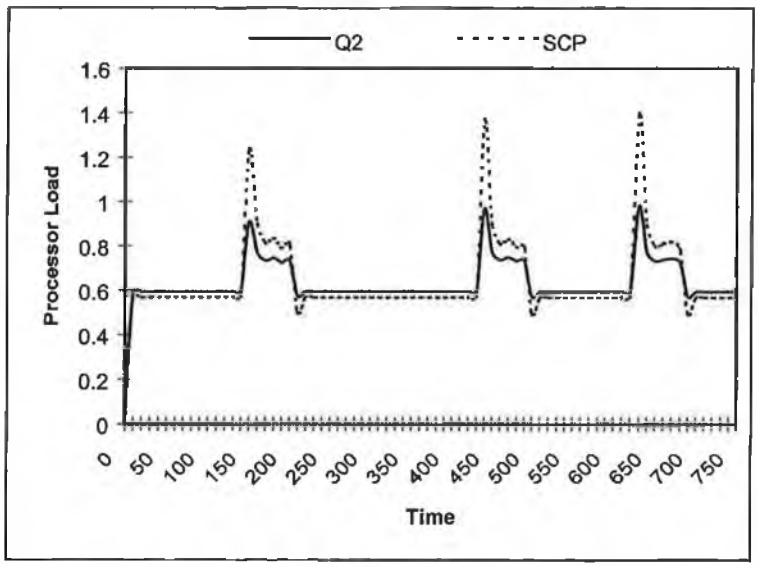


Fig. 5.21: SSP and SCP Processor Loads during Overload due to Bursty Traffic

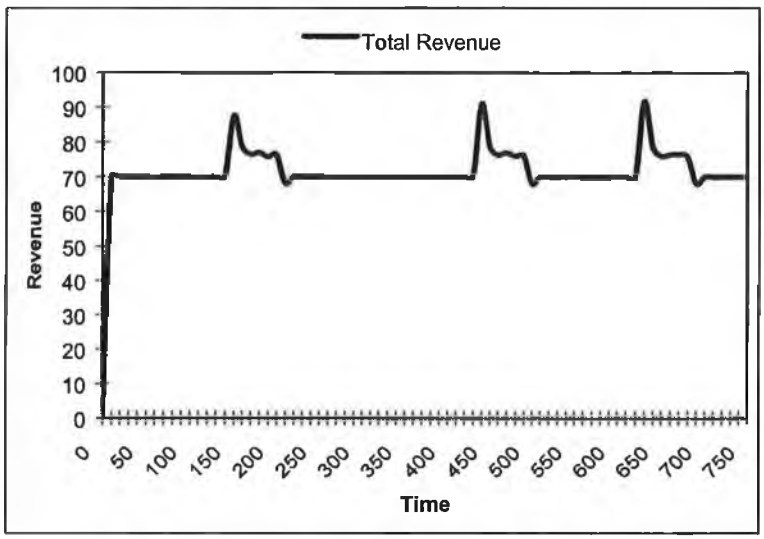


Fig. 5.22: Revenue during Overload due to Bursty Traffic

It was also interesting to note that, if SCP Constraint (iii) and SSP Constraint (iv) were amended so that $p_{a,i} \geq p_{a,j} \ \forall i > j$, the revenue remained equivalent, but in cases of SCP overload, rejection of local calls was forced (so that the advantage of rejecting only calls which caused the overload was lost), and therefore more televoting calls were accepted, resulting in greater load levels at the SCP (without exceeding the threshold) and greater service delays over all IN calls.

5.6 An Optional Extension to the Global IN Strategy – FDOC

It is interesting to note that the above IN congestion control strategy, by its nature, ensures that SCP congestion due to an excess of calls to focused destinations is unlikely. This is because it effectively allocates SCP or SSP capacity to different call types based on their weights and revenues, and therefore the only way that focused overload of the SCP could occur would be if the focused destination was of the call type with the highest weight and revenue in the network. As such, the enforcement of service fairness is implicit in the congestion control algorithm. However, the strategy still does not ensure fairness *within* an IN service type. It is still possible for calls to one destination to use an excessive amount of the allocated resource within the service type (if the arrival rates for that destination are much greater than arrival rates for other destinations). This is not necessarily undesirable, as it does not affect the performance or revenue of the network. However, in practical terms, the probability of successful completion of calls is greater for destinations with lower arrival rates as the probability that the destination node is overloaded is lower. Therefore, it may be more desirable to accept, within a service type, all calls to destinations with low arrival rates and selectively reject calls to destinations with high arrival rates in order to make the best use of resources allocated to a service type. This scenario is generally referred to as Focused Destination Overload Control (FDOC). However, in the case of our algorithm, this is a misnomer, as the IN congestion control algorithm, by its nature, prevents overload by a particular call type, so focused destination overload is extremely unlikely. Instead, this "FDOC" extension forces fairness within a given call type.

A simple optional extension to the IN congestion control algorithm described above facilitates FDOC (and was published in [Lodge98a]). At the SCP, the arrival rates to specific destinations within a service type should be monitored over the course of a control interval (in a manner similar to that described in [Rumsewicz95]). At the end of the control interval, resources within the SCP are allocated to service types as described above, resulting in the specification of a probability of acceptance $p^{a}_{SCP,j}$ for service type j. Therefore, the SCP resources available for calls of this type is $R^{avail}_{SCP,j} = \frac{p^{a}_{SCP,j} \lambda^{est}_{total,j}}{\mu_{SCP}}$. Resource requirements $R^{req}_{SCP,j,dn}$ to each destination within service type j may now be expressed based on the monitored arrival rates to each destination in the previous interval,

i.e. $R^{req}_{SCP,j,dn} = \frac{\lambda^{new}_{SCP,j,dn}}{\lambda^{new}_{SCP,i} p^{a}_{SCP,i}} R^{avail}_{SCP,j}$, where $\lambda^{new}_{SCP,j,dn}$ is the arrival rate at the SCP of calls of type j to destination dn during the previous interval. We may now use a simple method to allocate resources to calls to each destination by applying the following focus elimination function to all resource requirements $R^{req}_{SCP,j,dn}$:

$$R^{ass}_{SCP,j,dn} = f(R^{req}_{SCP,j,dn}) = \begin{cases} R^{req}_{SCP,j,dn} & , R^{req}_{SCP,j,dn} \le R_{Thr,j} \\ g(R^{req}_{SCP,j,1},\ldots,R^{req}_{SCP,j,N}) & , R^{req}_{SCP,j,dn} \ge R_{Thr,j} \end{cases}$$

where $R^{ass}_{SCP,j,dn}$ is the resource assigned to destination dn, $R_{Thr,j}$ is some threshold value for the SCP resource assigned to service type j and N is the number of destination numbers within the service type. The form of the function g and the value of $R_{Thr,j}$ should be chosen so that:

- The algorithm is kept simple, to reduce processing overheads,
- All available resources are assigned, i.e. $\sum_{dn} R^{ass}_{SCP,j,dn} = R^{avail}_{SCP,j}$,
- Destinations which require more resources are assigned more resources, i.e. if $R^{req}_{SCP,j,dn_x} < R^{req}_{SCP,j,dn_y}$, then $R^{ass}_{SCP,j,dn_x} < R^{ass}_{SCP,j,dn_y}$,
- Destinations with "small" resource requirements should be assigned all capacity required when possible.

A simple example of a non-linear function g would be:

$$R^{ass}_{SCP,j,dn_x} = g(R^{req}_{SCP,j,1},..,R^{req}_{SCP,j,N}) = R_{Thr,j} + \frac{R^{req}_{SCP,j,dn_x} - R_{Thr,j}}{\sum_{dn} (R^{req}_{SCP,j,dn} - R_{Thr,j})_+}$$

where $X_+ = 0$ if $X \le 0$ and $X_+ = X_+$ if $X > 0$.

Once the available resources have been assigned, the assignments may be translated into a percentage of available resource, and passed to the SSPs along with the new value of $p^{a}_{SCP,j}$ for that service type. The SSP algorithm then evaluates the appropriate $p^{a}_{n,j}$ and allocates this over all destinations based on the received percentages.

Note, however, that the inclusion of this extension to the control algorithm requires the monitoring of all destination numbers within each call type, and as such, renders the algorithm substantially more processor-hungry. Another drawback of using such an extension is the fact that the estimate that the SCP makes as to the total arrival rate of each service type to the system becomes less accurate, as it is now based on a very large number of parameters. This renders the entire algorithm less efficient overall. A conclusion of this is that the basic algorithm itself provides

perfectly adequate cover for the prevention of overload due to one call type, and therefore IN overload due to calls to a specific destination will generally not be permitted. Therefore, the only advantage of including the FDOC extension is that it enforces fairness within a given call type, and therefore, it should not be used unless this level of fairness is critical to a service provider.

5.7 Conclusions

The results for this control strategy are excellent. In all scenarios, with different traffic distributions and mixes bringing about overload, the congestion control algorithm causes the load of all elements to converge to the specified threshold very quickly and without oscillations. Therefore, the best possible use is made of all resources during overload. This is reflected in the fact that service delays remain consistently within required bounds. The strategy also provides the added bonus that revenue is optimised at all times, even during overload!

The weighting strategy used is defined to be extremely flexible so that it can encompass both functional and non-functional requirements of the service providers and users, and the results prove that, by including the weights as a constraint in the optimisation algorithm, the relative importance of calls is maintained at all times, even during extreme overload conditions.

Examining the requirements on the optimisation-based global IN congestion control strategy shows that it meets the basic requirement of protecting all network elements under all load conditions. It also meets the desired characteristics, in that the strategy is:

Scalable - all that is required to target the algorithm to a particular network resource is that a number of parameters, e.g. $e_{SCP,j}$, service rates and weights need to be defined. Also, the addition of new resources to the network (e.g. a new SSP) will not require alterations to the existing congestion control algorithms in other resources.

Flexible – the definition and allocation of call weights is at the discretion of the service provider, so that their needs with regard to service differentiation may be met. New constraints may be added to the optimisation algorithm – the only requirement is that they must be linear. Extensions to the algorithm such as the facility for FDOC may be added easily. The strategy therefore proves itself to be extremely flexible.

Fair – in this chapter, the optimisation-based strategy has proven itself to provide implicit service fairness, within the bounds of the priority system defined. Further, more detailed levels of fairness may be provided through the extension of the algorithm, e.g. to cater for fairness within a service type. Optimisation also exhibits subscriber fairness – while not explicitly shown in the results of

this chapter, it should be intuitively obvious as optimisation uses a PT throttle, which was proven in Chapter 4 Section 4.3.3 to be subscriber fair.

The only requirement that has not yet been proven to be met is that of efficiency – the resource requirements of the algorithm are as yet unclear. Obviously, they will be greater than for a simple algorithm like CCC, but it remains to be verified that these excess resource requirements are worth the cost, in terms of the value add provided by the strategy. In order to investigate this, the operation of the optimisation-based global IN congestion control strategy will be compared with classic CCC/CG, Window and a simpler dynamic IN congestion control strategy (devised for the purpose of comparison) in Chapter 6.

Chapter 6

Comparison between IN Congestion Control Strategies

6.1 Introduction

The optimisation strategy presented in Chapter 5 has been confirmed as meeting the basic requirements on a congestion control algorithm, namely it effectively protects all elements of the IN from congestion under all load levels and traffic mixes. It was also demonstrated that the algorithm is flexible, scalable and fair. We now need to show that the greater processor overheads required to use the strategy are worthwhile, in terms of the increased performance of the IN. To do this, we must compare the behaviour and resource requirements of the optimisation-based global IN congestion control strategy with that of other strategies. Chapter 4 showed that CCC provides the best performance among the existing, commonly used SCP congestion detection methods. It was also shown that the CG throttle responds faster to the onset of overload than PT, and therefore, while PT has a number of desirable characteristics, CG is more efficient at protecting the SCP during overload. The results for the Window strategy presented in Chapter 4 were inconclusive – it was seen to respond very quickly to the onset of overload, but was extremely inconsistent across variations in load levels. Therefore, to establish the superiority of the optimisation-based strategy, it should be compared with both classic CCC/CG and Window. However, this is not a truly fair comparison, as any classic IN control strategy will be based on fixed parameters which can never provide efficient congestion control for all traffic mixes, as they tend to be based on the assumption that either:

- All calls have the same load requirement, or
- If calls have different load requirements, then the ratio of arrivals for the different call types that comprise the total arrivals to the system is constant.

The result of this is that no fixed parameter values can be defined which apply to multiple call types with different load requirements and varying traffic mixes (as was verified in Chapter 4, Section 4.4). Therefore, optimisation (or in fact, any reasonable dynamic congestion control strategy) should automatically outperform all of the classic strategies. So, in order to rigorously evaluate the advantages and disadvantages of the optimisation strategy, it should also be compared with another dynamic strategy. To facilitate this, a new dynamic IN congestion control algorithm was specified. This is based on the use of a dynamic version of CCC in conjunction with a dynamic combined PT/CG throttle – i.e. a strategy that is both scalable and dynamic. Therefore, in this chapter, we will compare the optimisation-based global IN congestion control strategy with:

1. **Classic CCC/CG:** a classic IN overload control strategy, in which the SCP congestion control algorithm uses CCC to evaluate overload levels at the SCP and CG to throttle IN traffic at SSP Q2, and an independent SSP congestion control algorithm which consists of CCC at Q2 of the SSP setting CG throttles at Q1 to throttle all calls equally,

2. **A Window-based Strategy**: Window, located at the output of the SSP, prevents overload of the SCP, while SSP protection is provided by the same SSP CCC/CG algorithm used in the classic CCC/CG strategy,

3. **Dynamic CCC/CG**: a scalable and dynamic adaptation of classic CCC/CG.

The criteria we use to compare these strategies are SCP load, SSP load and throughput, network revenue and service delays. We also evaluate the cost efficiency of the algorithm in terms of both its processor requirements and the number of counters required to monitor the statistics required for the algorithm.

Section 6.2 describes the classic independent CCC/CG strategy, the Window strategy and the new dynamic CCC/CG strategy. Section 6.3 compares the operation of the three strategies under the stated criteria. Section 6.4 summarises the results and draws conclusions as to if and when the use of each of the strategies is most appropriate.

6.2 The Strategies used for Comparison

Three different strategies were used for IN congestion control to facilitate comparison in this work (as published in [Lodge99]). For all strategies, the SSP throttles used applied to all traffic types equally – i.e. when the SCP becomes overloaded, it requests that all IN calls be throttled equally in the SSPs, and when an SSP becomes overloaded, it throttles all incoming traffic (IN and non-IN) equally in order to protect itself.

The first IN overload control strategy (*classic CCC/CG*) is very simple and works as described in Chapter 4, Section 4.2. For SCP overload control, the total number of arriving calls is counted over an interval. At the end of the interval, a CCC algorithm at the SCP compares this count against a table to establish the level of overload. The overload level is returned to all SSPs, which look up a table to establish which CG throttle level should be applied at Q2 to restrict all IN calls. The SSP overload control strategy is very similar, with the CCC algorithm at Q2 sending overload levels to Q1, where CG throttles are put in place on all calls. Note that this strategy is similar to the independent IN congestion control strategy described in Chapter 5, Section 5.3.1. The CCC table and CG table parameter values were derived from the assumption that all IN call types had equal arrival rates at the SCP (but different load requirements) and that IN calls comprised 30% of total SSP traffic.

The second strategy (*Window-based*) protects the SCP as described in Chapter 4, Section 4.2.3. However, the use of Window for SSP congestion control is inappropriate, as it is only suitable for the protection of remote network elements. Therefore, another algorithm must be used for SSP

overload protection. For simplicity, the SSP overload control strategy used in classic CCC/CG (i.e. CCC at Q2 with CG at Q1) will therefore be used in conjunction with Window at the *ssf* process.

The third strategy (*dynamic CCC/CG*) operates in a similar manner to the classic CCC/CG strategy, but has slightly more complex algorithms. The steps of the algorithm are shown in Figure 6.1, and are described below.

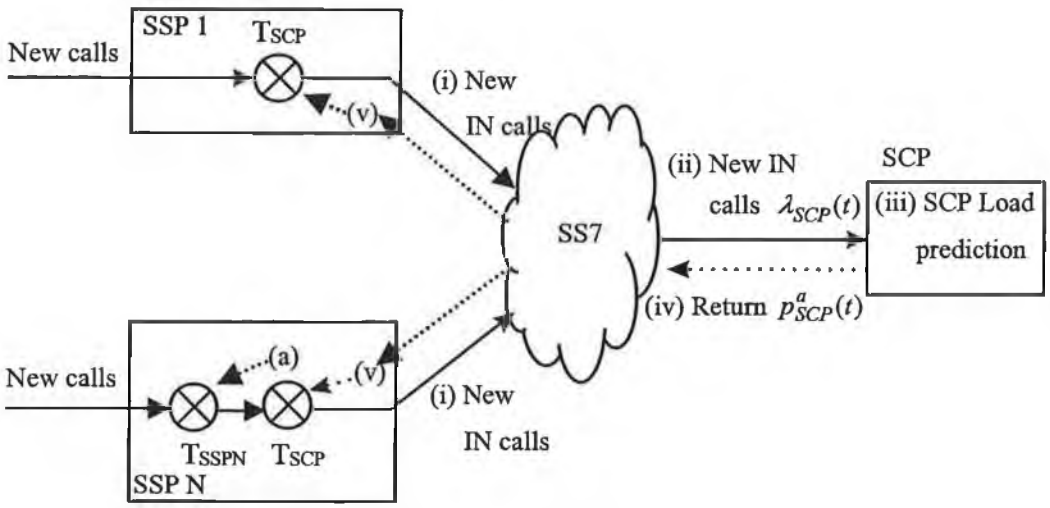


Fig. 6.1: The dynamic CCC/CG strategy

For SCP overload control, the total number of arriving calls for each IN service type is counted over an interval T (step (ii)). At the end of the interval, the CCC algorithm at the SCP then predicts what the total SCP load will be due to these calls (step (iii)), according to:

$$\rho^{pred}_{SCP}(t) = \frac{1}{\mu_{SCP}} \left(\sum_{j=2,3,5} \frac{e_{SCP,j} \lambda_{SCP,j}(t)}{p^a_{SCP}(t-T)} \right)$$

where $\rho^{pred}_{SCP}(t)$ is the predicted SCP load calculated at time t, $\lambda_{SCP,j}(t)$ is arrival rate of calls of type j to the SCP during the interval [t-T, t], μ_{SCP} is the service rate of the SCP, $p^a_{SCP}(t-T)$ is the value of the throttle put in place (at the SSP) by the SCP at the end of the previous interval and $e_{SCP,j}$ is the number of times a service request of type j will receive processing at the SCP during the course of its execution. Then, if this predicted load is less than the SCP threshold, the probability of acceptance at the SCP, $p^a_{SCP}(t)$, is set to 1.0. If it is greater than the threshold, $p^a_{SCP}(t) = Thr_{SCP} / \rho^{pred}_{SCP}(t)$ where Thr_{SCP} is the defined SCP threshold. Note that $p^a_{SCP}(t)$ is a single probability of acceptance and applies to all IN calls equally. If the resultant value of $p^a_{SCP}(t)$ is different to that defined at the end of the previous interval, it is sent to all SSPs, either in CG messages, or encapsulated in service-related messages (step (iv)). When SSP_n receives the message, it converts $p^a_{SCP}(t)$ to an IN CG gap interval $G_{IN}(t)$ (step (v)) using the simple formula:

$$G_{IN}(t) = \frac{1 - p^a_{SCP}(t)}{\lambda_{n,IN}(t) p^a_{SCP}(t)}$$

where $\lambda_{n,IN}(t)$ is the arrival rate of IN calls at SSP_n during the interval [t-T, t] and all other parameters are as previously defined. Once the gap interval of the CG throttle is evaluated using this formula, it put in place on all IN calls at Q2.

The SSP overload control algorithm is slightly different, in that it is based on the use of a dynamic LMC algorithm at Q2 of each SSP. LMC is used here because the mean load at Q2 over the duration of a monitoring interval reflects the value of the throttle put in place at Q2 by the SCP at the start of that interval, and therefore dynamic CCC/CG, while still an independent control strategy does at least partially take the (previous) state of the SCP into account. Note however that dynamic CCC/CG still fails to take the current state of the SCP into account. The dynamic LMC algorithm at Q2 (step (a)) estimates what Q2 load would be if no throttle were in place at Q1 as:

$$\rho^{est}_{Q2_n}(t) = \frac{\rho_{Q2_n}(t)}{P^a_n(t-T)}$$

where $\rho_{Q2_n}(t)$ is the actual load of Q2 of SSP_n at time t and $P^a_n(t-T)$ is the probability of acceptance for all calls (i.e. the throttle value) put in place at Q1 at the end of the previous interval. The new throttle parameter $P^a_n(t)$ is then calculated from $P^a_n(t) = Thr_{Q2_n} / \rho^{est}_{Q2_n}(t)$ (where Thr_{Q2_n} is the load threshold of Q2 of SSP_n) and sent to Q1, where the new gap interval $G_n(t)$ is calculated as:

$$G_n(t) = \frac{1 - P^a_n(t)}{\lambda_n(t) P^a_n(t)}$$

where $\lambda_n(t)$ is the mean of the total arrival rate to SSP_n during the interval [t-T, t] and all other parameters are as previously defined. Once the new gap interval has been derived, it is put in place at Q1 and applied to all new call arrivals equally.

6.3 Results of Comparison

In this section, we present the results for each of the given strategies. The behaviour of the strategies presented above is compared for five different load scenarios – namely, stationary (section 6.3.1), SCP overload (section 6.3.2), SSP overload (section 6.3.3), general overload (section 6.3.4) and overload due to bursty traffic (section 6.3.5). In all cases, the SCP's and SSPs' service rates remain the same (to ensure a fair comparison), with the IN acceptance time at each

SSP Q2 set as a factor of 2.5 greater than the non-IN acceptance time and reject rates at both SSP Q1 and Q2 set considerably higher than all acceptance rates (as described in Chapter 5, section 5.2.1). Also, the load threshold defined for all physical elements during all simulations is 0.8.

6.3.1 Scenario 1: Stationary Behaviour

Here, constant (different) arrival rates are applied to each of the SSPs. The traffic mix is such that the load of SSP1, SSP2 and SSP3 are over the defined threshold, while the SCP is overloaded (due primarily to televoting requests). This is shown in Figure 6.2, where the load applied to each element is expressed relative to the capacity of that element. Note that this constitutes a general overload (i.e. multiple physical elements overloaded simultaneously).

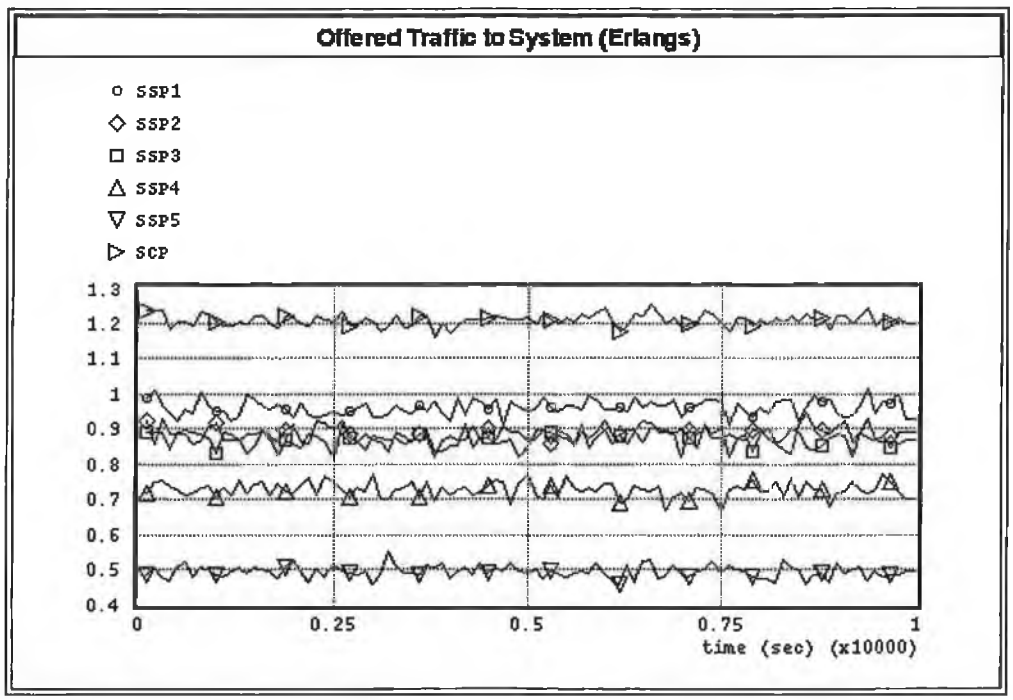


Fig. 6.2: Offered traffic for stationary case

The resultant SCP load for each of the four applied strategies is shown in Figure 6.3. Here it may be seen that only Window reacts quickly enough to the instantaneous onset of overload to ensure that the SCP load never exceeds the threshold. However, as the Window timer duration is fixed and less than the average response delay for televoting, Window tends to overprotect the SCP, maintaining the load at a mean of 0.73, resulting in the unnecessary rejection of many calls. All other strategies, on the other hand, allow the SCP to overload and the SCP queue to build up during the first monitoring interval of the simulation. Static CCC/CG never recovers from this, primarily due to the fact that it does not account for the fact that televoting calls require more processing at the SCP than do freephone calls – in fact, the CCC algorithm at the SCP only detects that a small overload has taken place and the resultant minimal throttles put in place at Q2 of each

SSP are ineffectual. Dynamic CCC/CG and optimisation, however, do estimate the overload level correctly at the end of the first monitoring interval, and immediately put throttles in place that, over the course of the next few intervals, reduce the SCP queue length and load to the defined threshold. Both strategies then maintain the load at this threshold, with optimisation experiencing smaller oscillations than dynamic CCC/CG.

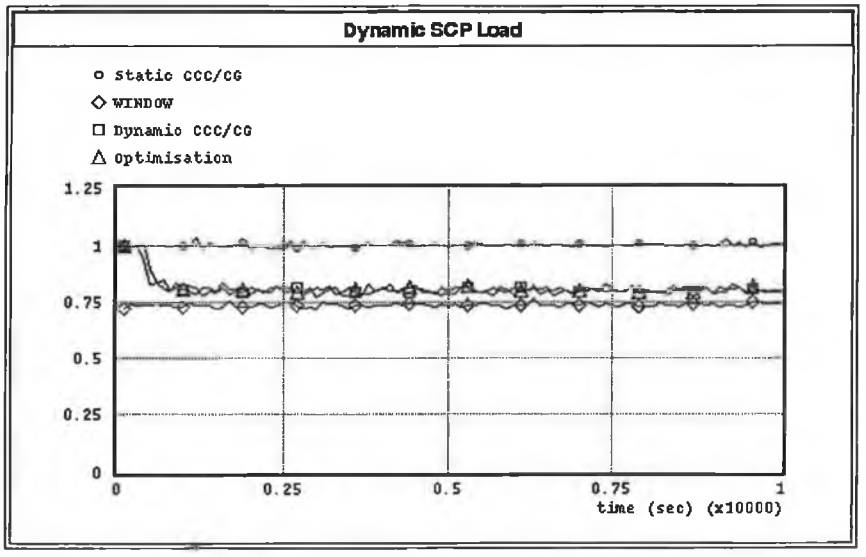


Fig. 6.3: SCP load for stationary case

The corresponding load of SSP1 is shown in Figure 6.4.

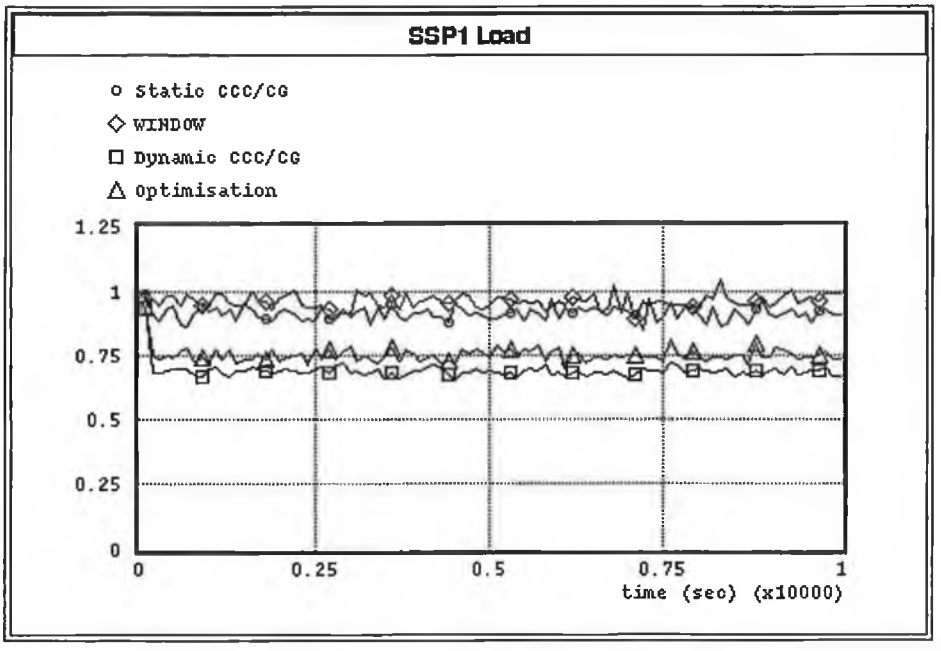


Fig. 6.4: SSP1 load for stationary case

Note that the Window-based strategy provides the worst performance in the SSP. The behaviour of static CCC/CG, while not acceptable, is better than Window. This may at first seem incongruous, as both strategies use exactly the same static CCC/CG algorithm for SSP protection. However, the

different locations of the SCP throttles account for the different behaviour in the SSP. Window is located at the output of the SSP (i.e. at the *ssf*) and therefore all calls receive full processing at Q2 before being throttled by Window. Static CCC/CG places its SCP throttles at Q2, with the result that some calls are rejected here (with rejection having lower processing overheads than acceptance), thus keeping Q2 load lower than that for Window. However, in both cases, the SSP static CCC/CG algorithm fails to detect SSP overload as it does not differentiate between IN and non-IN calls (and their different load requirements at Q2). The result, therefore, for both strategies is that the SSP remains in a permanent state of overload. Dynamic CCC/CG and optimisation, on the other hand, detect overload at both SCP and SSP at the end of the first monitoring interval, and put the correct throttles in place to protect both elements. The result of this is that SSP load (after the initial overload due to the monitoring delay associated with both strategies) remains well below the SSP threshold.

Through observation of the SCP and SSP loads, it may be concluded that neither Window nor static CCC/CG perform well at protecting all IN physical elements – Window protects the SCP but fails to protect the SSPs, while static CCC/CG allows all elements to overload. Only dynamic CCC/CG and optimisation succeed in protecting the SCP and SSPs simultaneously from overload. In fact, both strategies maintain the mean loads of the various elements at approximately the same value. For effectiveness, therefore, the strategies are equivalent. Regarding speed of convergence, dynamic CCC/CG reacts slightly faster than optimisation – this is because, as described in Chapter 4, Section 4.3.3, CG reacts faster to the onset of congestion than PT (the throttle used by optimisation). Regarding operation of the algorithms, Figure 6.5 shows that, during the initial SSP overload, dynamic CCC/CG rejects all calls equally at first (at Q1), until SCP throttles become effective (at Q2) and it becomes unnecessary to reject non-IN calls, from which point only IN requests are throttled (equally and at Q2). For optimisation, at no point are non-IN calls rejected – the optimisation algorithms balance the states of the SCP and SSP and recognises that, by putting throttles on IN requests at Q2 to protect the SCP, the SSP overload situation will be automatically relieved. The result is that, at the start, televoting requests (the cause of the overload) are rejected at Q2 and then the acceptance rates of televoting, freephone and international freephone decrease according to their relative importance. The result is two-fold. Firstly, due to selective throttling, optimisation accepts more calls (i.e. calls with low load requirements such as non-IN, freephone and international freephone) and secondly, the overall revenue gained during the simulation is considerably greater for optimisation, as may be viewed in Figure 6.6 (showing revenue gained per second, in Irish pounds).

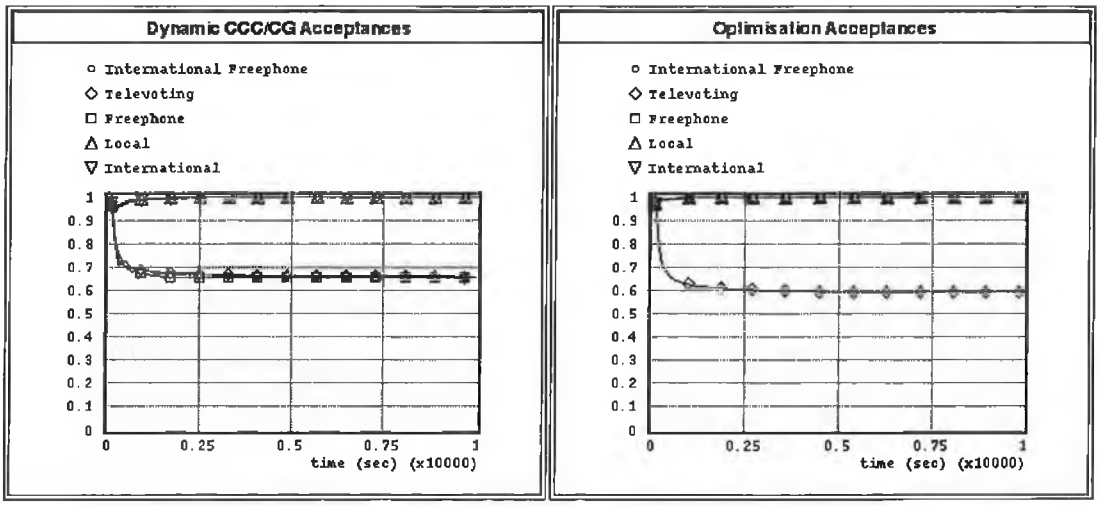


Fig. 6.5: SSP1 acceptances for stationary case

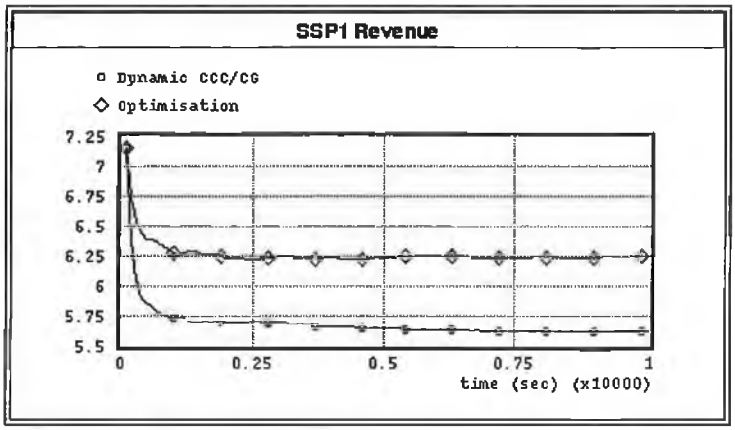


Fig. 6.6: SSP1 revenue for stationary case

To conclude, even in the stationary case, the behaviour of static CCC/CG and Window are unacceptable, in that they are incapable of maintaining the load of all elements in the IN at the threshold during overload. This is primarily due to the fact that both strategies are based on the use of fixed parameters – static CCC/CG underprotects its resources when overload is caused by requests with greater load requirements (and overprotects them when overload is caused by requests with low load requirements), while Window overprotects its resources when subjected to requests with high processing requirements (and therefore greater delays). Optimisation and dynamic CCC/CG provide much better results, keeping all elements at their threshold. However, optimisation has the advantages of providing higher call acceptance rates (during SCP overload) and network revenues.

Due to the unacceptable behaviour of static CCC/CG even under constant arrival rates to the system, it will no longer be considered in this chapter. Window, however, does have the advantage of speed of reaction and is effective at protecting the SCP, so its behaviour under different load conditions will continued to be examined and compared to the dynamic CCC/CG and optimisation strategies.

6.3.2 Scenario 2: SCP Overload

In order to achieve SCP overload, we vary the offered load and traffic mix to each SSP by increasing the arrival rates of televoting and freephone requests at all SSPs linearly as shown in Figure 6.7 for SSP1. The resultant total arrival rate to each element in the network is shown in Figure 6.8 – note that the offered traffic at each element is expressed in terms of the capacity of that element. Note also that, while the offered traffic to a number of SSPs is, at various stages, greater than the capacity of those SSPs, the SCP is the first element to become overloaded. Therefore, if the congestion control strategies respond correctly and quickly to the SCP overload, the rejection of IN calls at the SSPs should ensure that no SSP ever experiences congestion.

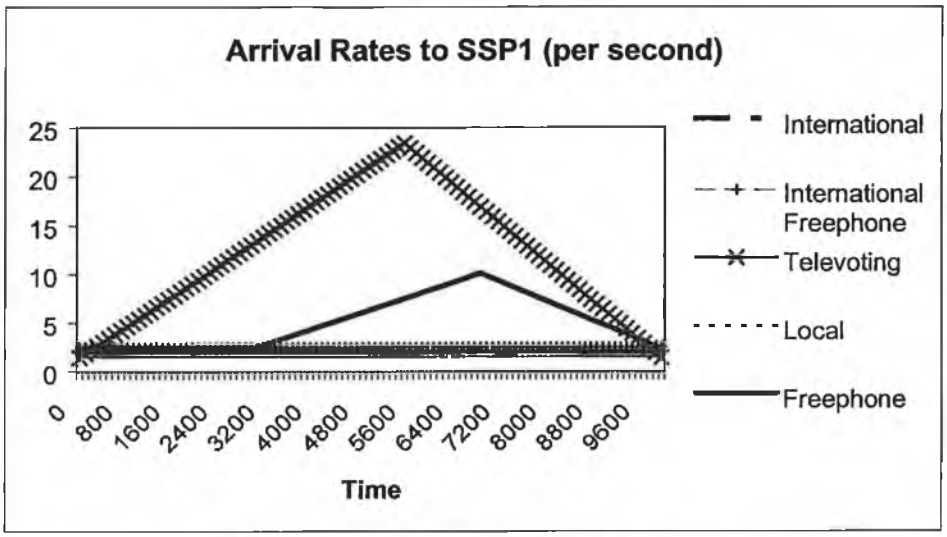


Fig. 6.7: Arrival rates to SSP1 for SCP overload

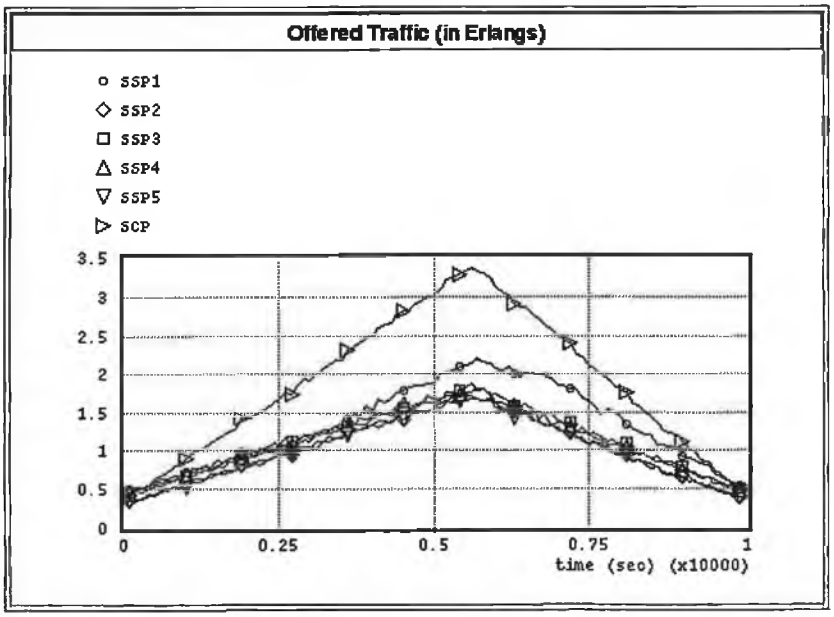


Fig. 6.8: Total offered traffic to all IN physical elements

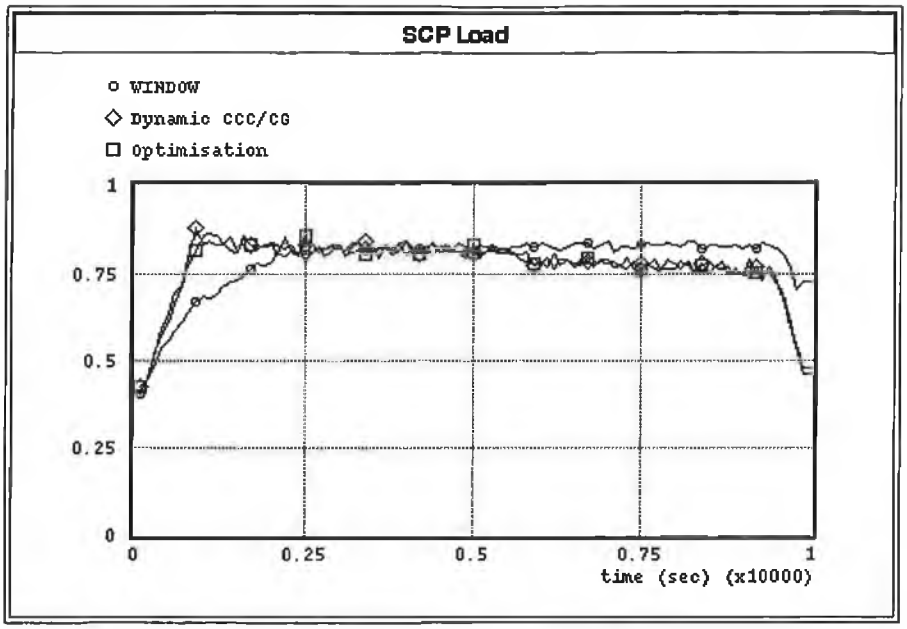


Fig. 6.9: SCP load for SCP overload

Figure 6.9 depicts the resultant SCP load for the Window, dynamic CCC/CG and optimisation strategies. Examining Window first, it may be seen that this strategy initially overprotects the SCP – this is due to the fact that televoting delays may be greater than the Window timer duration even when no overload exists. Therefore, Window starts to reject calls before the SCP load reaches 0.8 SCP Erlangs. However, the SCP load does eventually converge to a constant value of approximately 0.835 and remains there until the applied load becomes less than 1 SCP Erlang.

The behaviour of dynamic CCC/CG and optimisation, on the other hand, is quite different. Both strategies do not put controls in place until the SCP threshold is reached and therefore, there is no unnecessary rejection of calls. After this point, the SCP load is maintained at an average of 0.815 Erlangs until the applied traffic rate reaches its peak. The reason neither strategy maintains the load at exactly 0.8 is that, each time each strategy evaluates the overload situation, it does so based on existing traffic measurements and therefore puts corresponding throttles in place to maintain the load at exactly 0.8. Therefore, when the applied traffic increases during the next monitoring interval, the throttles accept more calls than had been expected and the resultant load is slightly greater than the threshold. A similar situation arises when the applied traffic levels are decreasing – neither strategy predicts the downward trend in traffic and therefore causes too many calls to be rejected, resulting in mean SCP loads of approximately 0.785.

The SSP Q2 loads for the same scenario are shown in Figure 6.10. Note that Window fails to protect the SSP from overload – this is for two reasons, the first being that the CCC/CG strategy at Q1 fails to detect overload due to IN traffic, as it does not take into account the greater load requirements of IN calls at Q2, while the second reason is that Window does not reject any IN

calls until after they have completed processing at Q2, so that much SSP resource is applied to processing calls which are then rejected by Window at the *ssf*. The behaviour of dynamic CCC/CG and optimisation, however, is as expected – in protecting the SCP by throttling calls at Q2, the SSP is implicitly protected.

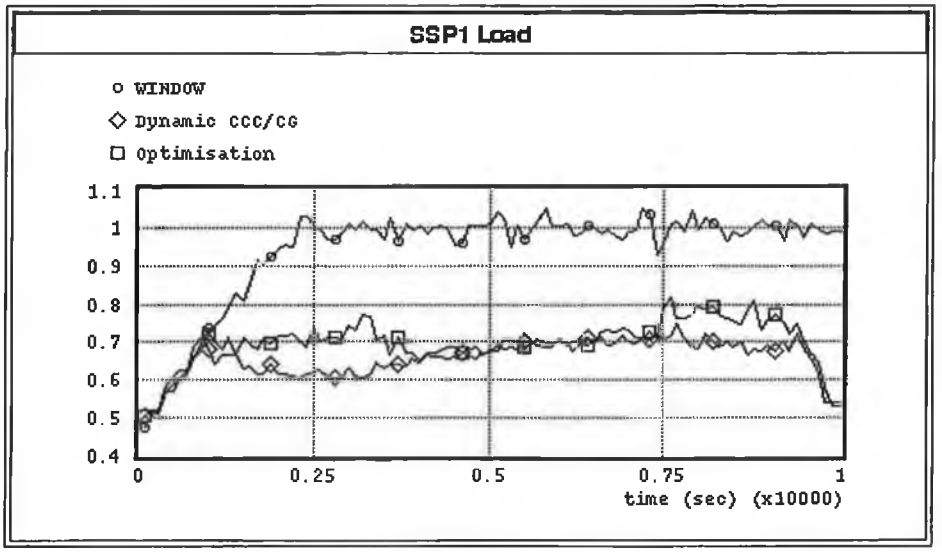


Fig. 6.10: SSP1 load for SCP overload

Note however, that the SSP load is consistently higher for optimisation than it is for dynamic CCC/CG. The reason for this may be observed in Figure 6.11, where it is shown that dynamic CCC/CG applies the same throttles to all IN call types, resulting in similar proportions of each IN call type being accepted. Optimisation, on the other hand, rejects call types selectively based on their weights, and by extension, their SCP load requirements. Therefore, televoting calls, which have the greatest SCP load requirement, are more strictly throttled than they are by dynamic CCC/CG. This means that more freephone and international freephone calls are accepted at Q2, resulting both in greater Q2 loads (and throughputs) and in greater overall numbers of accepted calls – optimisation accepts 5.35% more of the offered calls than does dynamic CCC/CG.

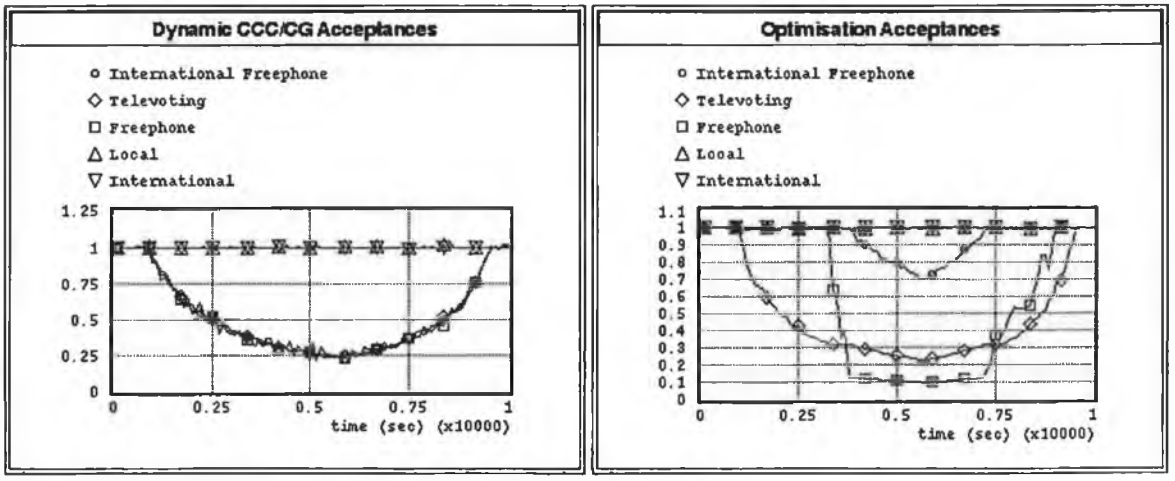


Fig. 6.11: Acceptances at SSP1 for SCP overload

There are two main implications of the selective throttling of optimisation. Figure 6.12 shows the revenue gained from the successfully completed calls at SSP1 – note that SSP1 revenue (and therefore network revenue) provided by optimisation is far greater than that provided by dynamic CCC/CG – this is because more calls (and calls of greater value) are consistently accepted by optimisation. Figure 6.13 shows the post-dialling delays experienced by various service types in the network. Note that, in all cases, dynamic CCC/CG delays are slightly less than optimisation delays. This is a direct result of the fact that optimisation causes the acceptance of more calls, resulting in slightly longer queue lengths and delays. Note however, that all delays experienced by services subjected to optimisation control are still well within acceptable bounds, as defined by [E.721].

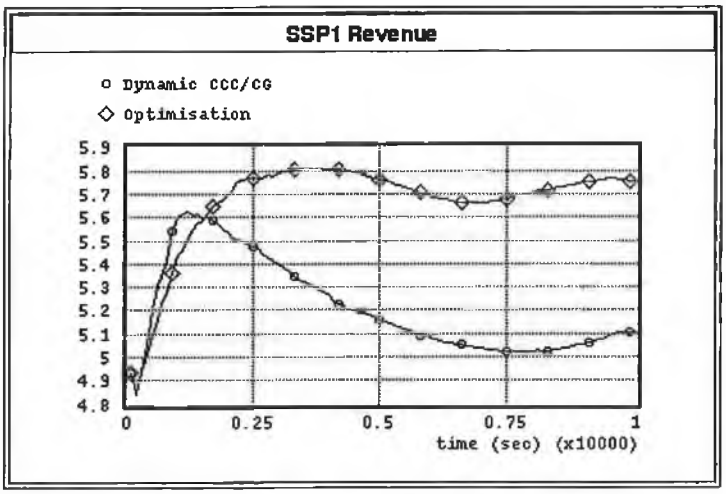


Fig. 6.12: Revenue of SSP1 for SCP overload

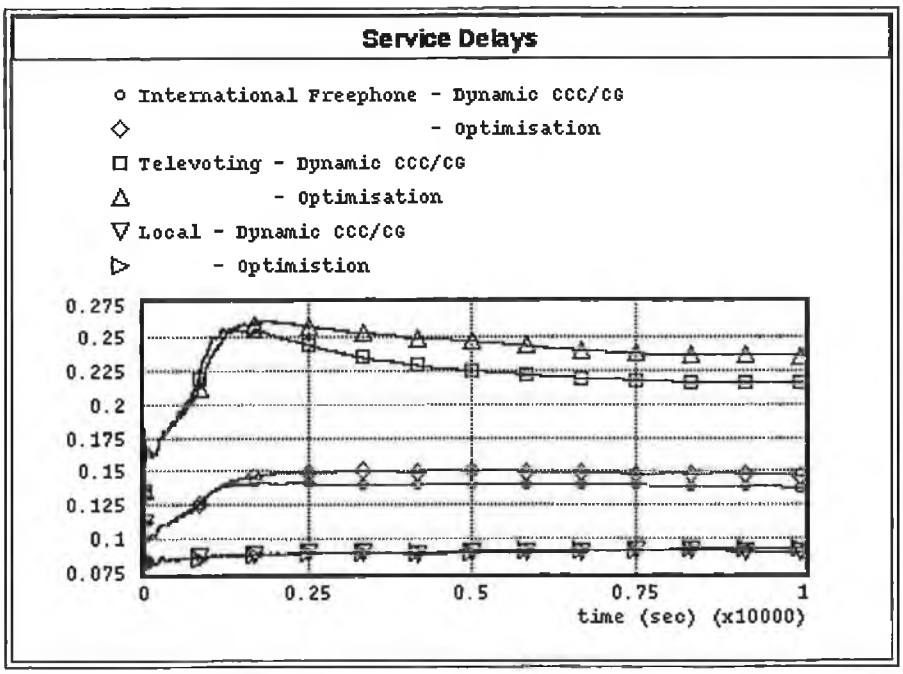


Fig. 6.13: Service delays at SSP1 for SCP overload

To summarise the results for this scenario, Window succeeds very well at protecting the SCP, but fails to protect the SSPs, while both dynamic CCC/CG and optimisation are equally efficient at protecting all elements at all times. However, optimisation, through selective throttling, manages to accept both more calls overall and more calls of greater worth, resulting in greater overall IN throughput and network revenue.

6.3.3 Scenario 3: SSP Overload

Here, an overload of SSP1 is invoked by increasing the arrival rates of international and local calls linearly as shown in Figure 6.14. The resultant total arrival rate to each element in the network is shown in Figure 6.15 – note that the offered traffic at each element is expressed in terms of the capacity of that element.

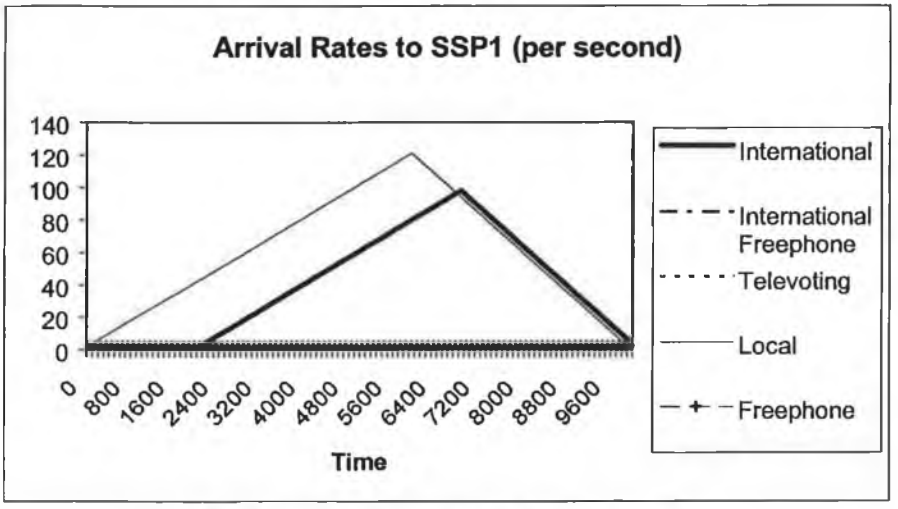


Fig. 6.14: Arrival rates to SSP1 for SSP overload

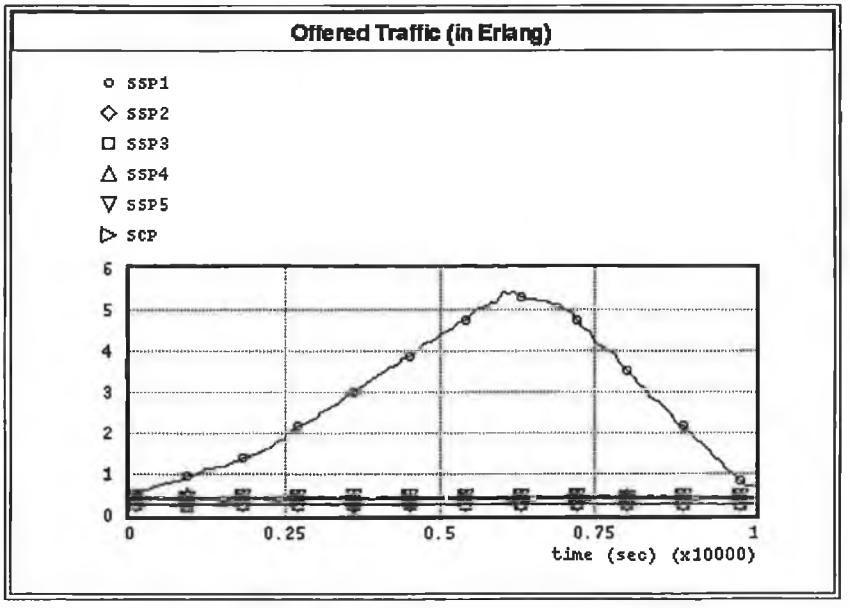


Fig. 6.15: Arrival rates for all IN physical elements

The resultant SSP1 Q2 load for each of the test strategies under this applied traffic is shown in Figure 6.16. Note first that the SSP's CCC/CG part of the Window strategy considerably overprotects that element. This is because this strategy bases its estimation of overload on the number of arriving calls and does not take into account the fact that the large number of non-IN arrivals have low SSP processing requirements. Therefore the throttles put in place by the Window-based strategy are excessively strict, with the result that the load of SSP1 Q2 is very low. Both dynamic CCC/CG and optimisation, on the other hand, put the correct throttles in place on detection of overload and therefore, after a period of convergence, keep the load at approximately 0.8 – again, with optimisation experiencing smaller oscillations than dynamic CCC/CG.

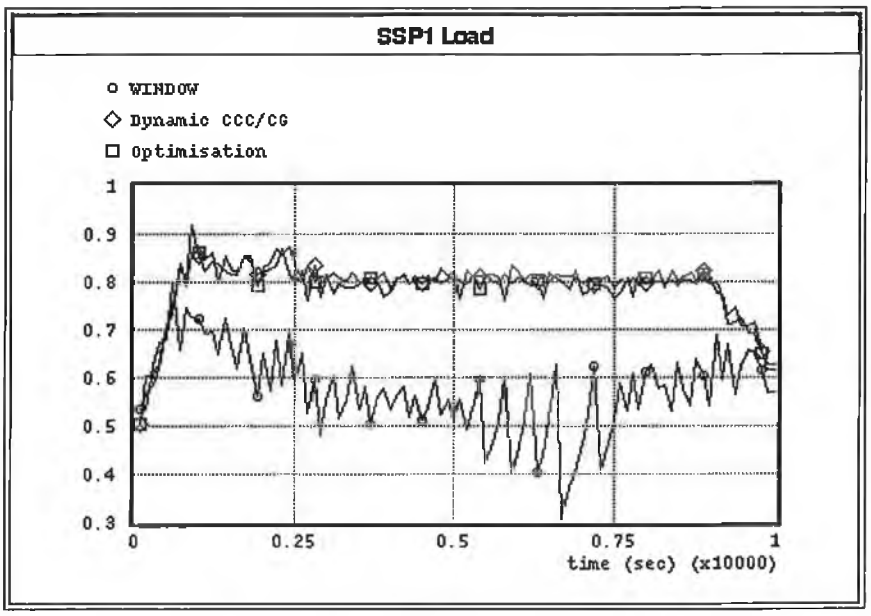


Fig. 6.16: SSP1 load for SSP overload

The resultant SCP load is shown in Figure 6.17 below. The load values for each of the strategies are low, mostly due to the low number of IN arrivals at the system and partially due to the rejection of calls at SSP1, and all traces are quite similar to each other. However, dynamic CCC/CG consistently provides the highest mean load values, with Window providing lower mean values and optimisation producing the lowest. The reason for this is similar to that described for the SCP overload scenario described in the previous section – i.e. dynamic CCC/CG applies the same throttles to all call types at Q1 of the SSP, resulting in similar proportions of each call type being accepted. Optimisation, on the other hand, rejects call types at Q2 selectively based on their weights. Therefore, as shown in Figure 6.18, more freephone, local and televoting calls (i.e. those calls with low weights) are throttled by optimisation than are by dynamic CCC/CG, while fewer international and international freephone are rejected. The result of this selective throttling is that optimisation, while producing the same SSP load levels as dynamic CCC/CG, gives consistently

lower SCP loads (primarily due to the lower televoting acceptance rate, as televoting has the greatest SCP load requirement).

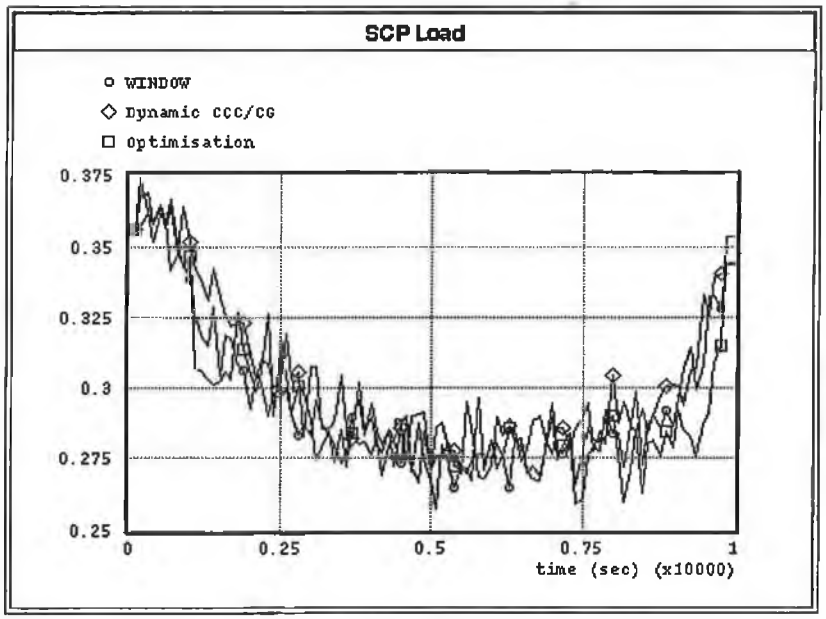


Fig. 6.17: SCP load for SSP overload

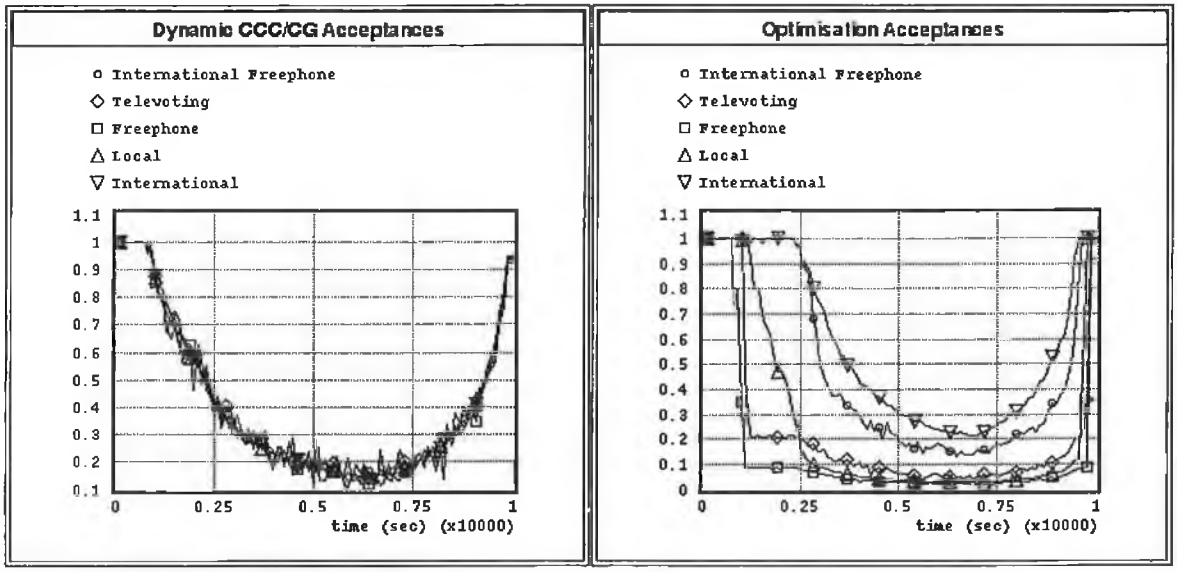


Fig. 6.18: SSP1 acceptances for SSP overload

The above graphs show that optimisation and dynamic CCC/CG are equally effective at protecting their resources, and far superior to Window. However, as may be seen in Figure 6.19, dynamic CCC/CG actually provides much greater SSP throughput than optimisation during SSP overload. The reason for this is that dynamic CCC/CG rejects calls at Q1 upon detection of SSP overload, while optimisation does not reject any calls until differentiation between call types becomes possible – i.e. at Q2. This means that, for optimisation, much SSP capacity is spent accepting calls at Q1 that are then rejected at Q2, whereas dynamic CCC/CG, having rejected all calls at Q1, uses all Q2 load in the acceptance of calls. The result is that dynamic CCC/CG provides much greater

SSP throughput – in terms of call acceptance rates, dynamic CCC/CG accepts 4.9% more of the offered calls during SSP overload than does optimisation (this figure may seem lower than expected, but may be accounted for by the fact that dynamic CCC/CG does not take load requirements into account when accepting call requests and therefore processes more (high load) IN calls than does optimisation).

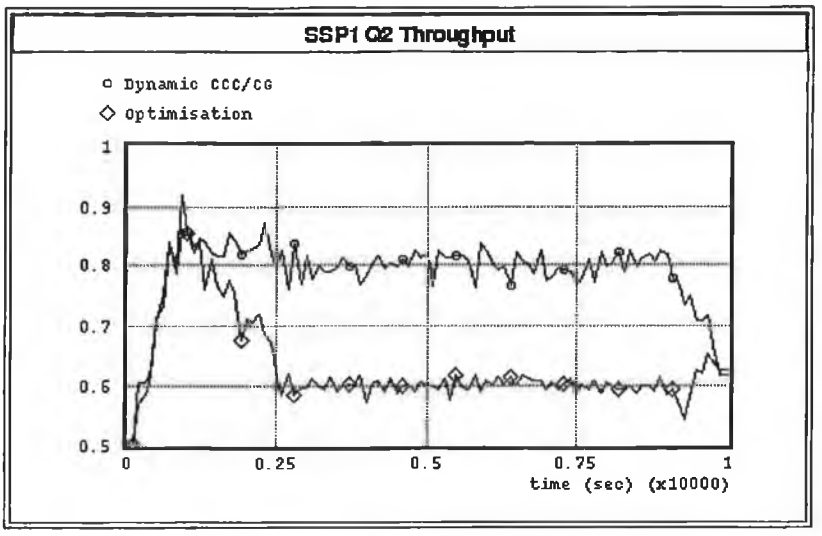


Fig. 6.19: SSP1 throughput for SSP overload

The conclusion of this is that both dynamic CCC/CG and optimisation are more effective strategies for SSP protection than Window – further, dynamic CCC/CG is more efficient for SSP protection than optimisation, as it provides equivalent protection but accepts far more calls. The only advantage of optimisation in this scenario is that even when its throughput is so much lower than that provided by dynamic CCC/CG, the fact that optimisation prioritises those calls with the greatest revenue means that this strategy still provides better revenue gain than either of the other two strategies, as may be seen in Figure 6.20.

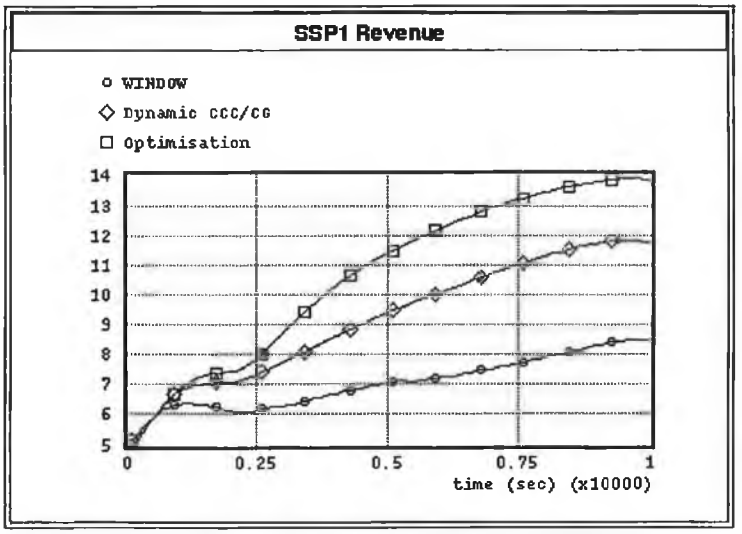


Fig. 6.20: SSP1 revenue for SSP overload

6.3.4 Scenario 4: General Overload

Here, overloads of the SCP and SSP1 are invoked by increasing the arrival rates of televoting at all SSPs linearly and the arrival rates at SSP1 for televoting, international and local calls linearly at different times, as shown in Figure 6.21. The resultant total arrival rate to each element in the network is shown in Figure 6.22 – note that the offered traffic at each element is expressed in terms of the capacity of that element and that the SCP becomes overloaded prior to SSP1.

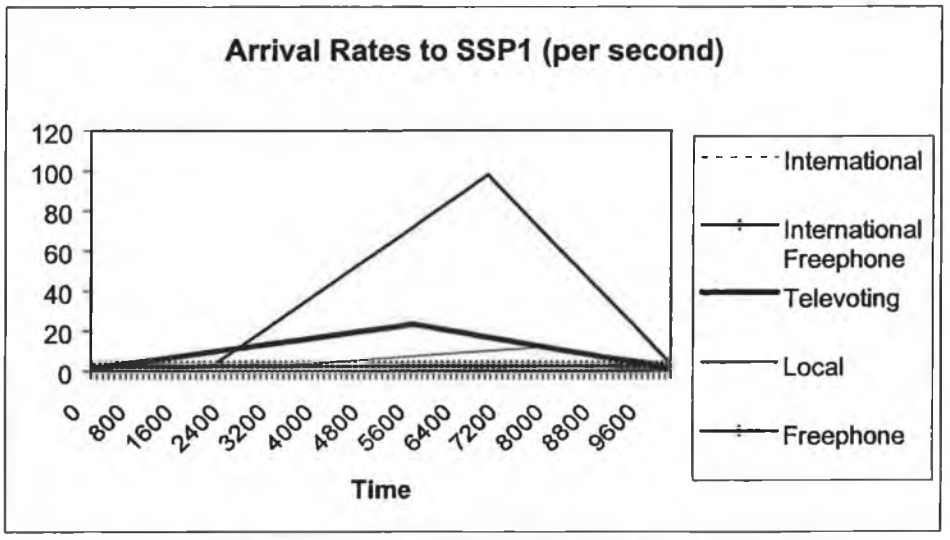


Fig. 6.21: Arrival rates to SSP1 for general overload

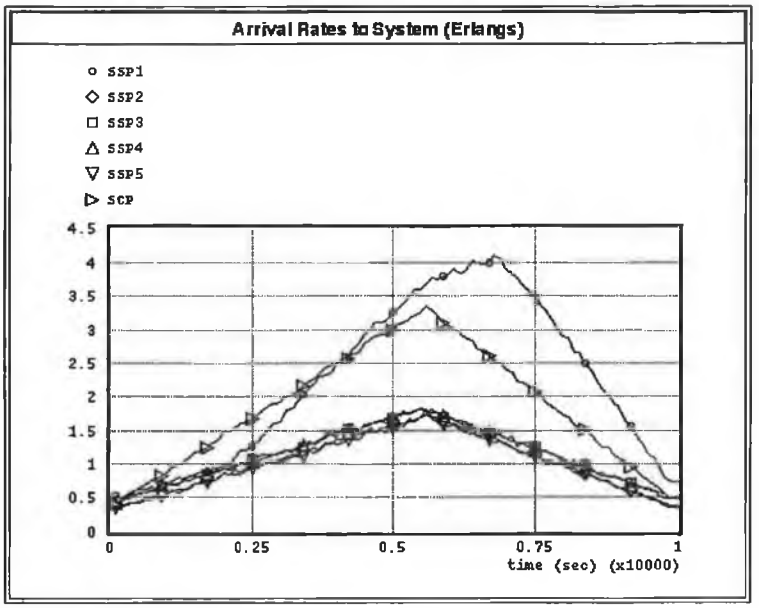


Fig. 6.22: Arrival rates for all IN physical elements

The resultant SCP load for each of the strategies being compared is shown in Figure 6.23, while the load of SSP1 Q2 is shown in Figure 6.24. By viewing both graphs, it may be observed that for the first part of the simulation, when correct response to SCP overload should be sufficient to protect the SSPs, Window still allows SSP1 to become and remain overloaded (because so much

SSP resource must be allocated to processing calls to the point where Window may throttle them – i.e. at the output of SSP1). At the same time, Window initially overprotects the SCP, and then gradually brings the SCP load to converge to a mean of approximately 0.8, where it remains until the SSP part of the Window strategy begins to respond to overload due to local calls at SSP1. This results in the rejection of an excessive numbers of calls (in particular televoting and local) at SSP1 Q1, thus bringing the mean SCP load down to about 0.76.

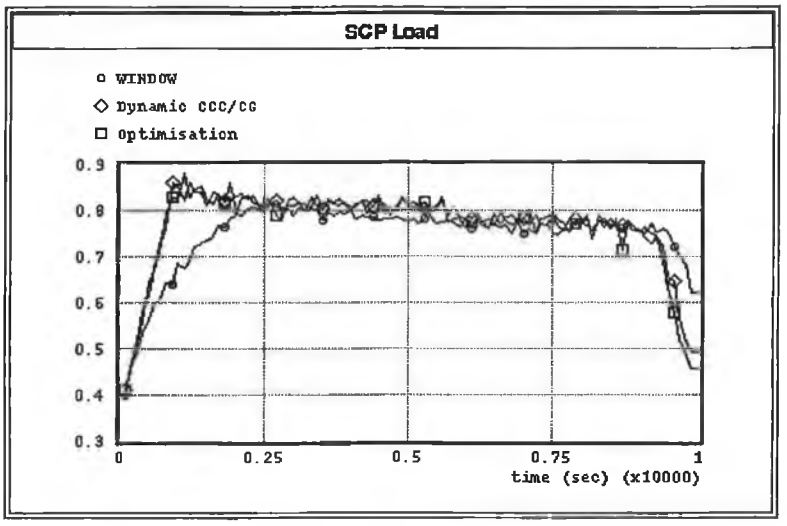


Fig. 6.23: SCP load for general overload

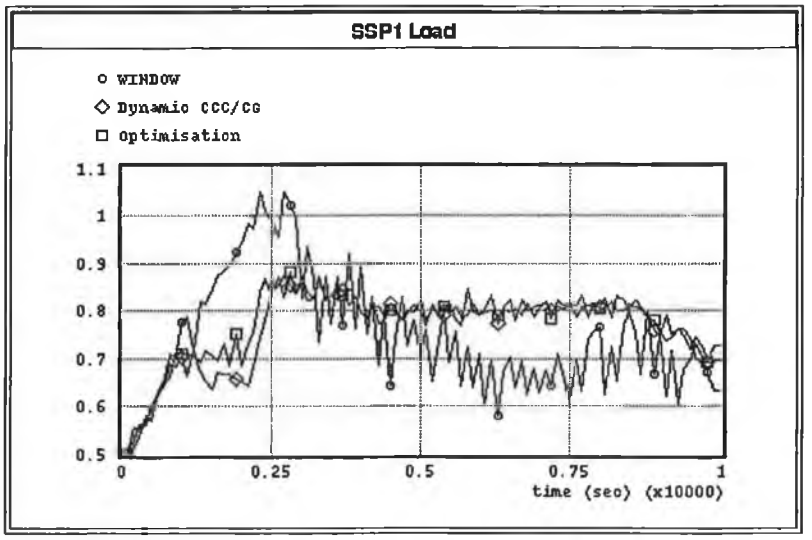


Fig. 6.24: SSP1 load for general overload

The operation of dynamic CCC/CG and optimisation at the SCP, on the other hand, is quite different. After an initial monitoring delay they both converge to approximately 0.81 SCP Erlangs (not exactly 0.8, because traffic arrival rates are increasing) until traffic rates begin to decrease, at which stage both strategies cause the SCP load to converge to a mean of approximately 0.785. This behaviour is identical to that portrayed in section 6.3.2 for the SCP overload scenario. Note that, unlike Window, the SCP load is not affected by the state of the SSP when dynamic CCC/CG or

optimisation is used. The reason for this may be seen in Figures 6.24 and 6.25. The SSP load curves show that, during the early stage of the simulation, the correct response by dynamic CCC/CG and optimisation to the SCP overload situation prevents SSP overload. Optimisation achieves this by throttling televoting calls only, while dynamic CCC/CG throttles all IN call types. Then later, as the local call arrival rate increases to the point where SCP congestion controls are insufficient to protect the SSP, optimisation balances the current states of both the SCP and SSP to devise the required rejection rates to protect both elements and therefore begins to reject local and freephone calls. Dynamic CCC/CG reacts differently – it balances the effects of the SCP throttles (which have been in place at Q2 during the previous monitoring interval) on Q2 load with the predicted effect of the total number of arriving calls and gradually increases the throttles on all call types at SSP1 Q1 accordingly (i.e. dynamic CCC/CG balances the *current* SSP state with the *previous* SCP state). Therefore, using different methods, both dynamic CCC/CG and optimisation put the correct controls in place to protect the SSP, without affecting the SCP load.

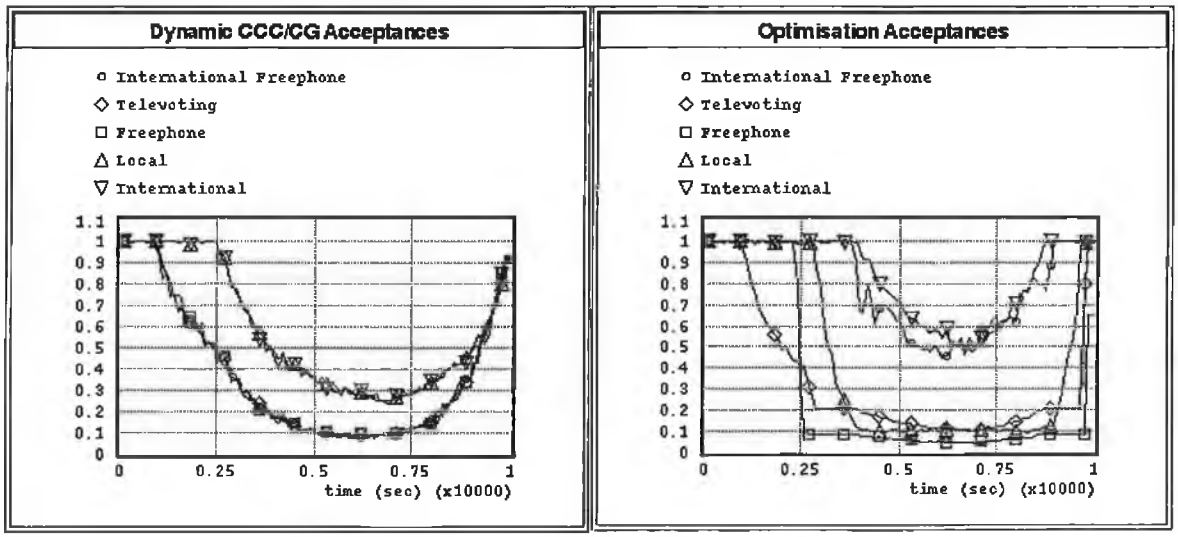


Fig. 6.25: SSP1 acceptances for general overload

Note that, as would be expected from sections 6.3.2 and 6.3.3 and as is shown in Figure 6.26, the SSP throughput is about 0.07 Erlangs greater for optimisation when the SCP is more overloaded than the SSP (and optimisation accepts 5.5% more calls than does dynamic CCC/CG), but in the inverse scenario, the fact that optimisation must expend extra resources accepting all calls in Q1 before being able to throttle them in Q2 means that the throughput of dynamic CCC/CG is 0.2 Erlangs greater than that for optimisation (and dynamic CCC/CG accepts 8.3% more calls).

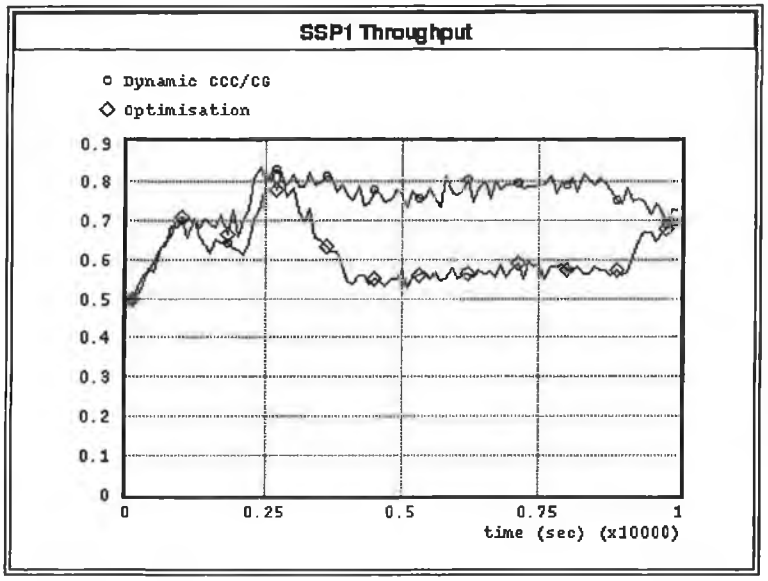


Fig. 6.26: SSP1 throughput for general overload

As a final comment on the general overload scenario, the revenue gained by SSP1 over the course of the simulation is shown for all three strategies in Figure 6.27. Note that Window is artificially high during the period of SCP overload – this is due to the fact that the SSP is underprotected at this time and is therefore accepting an unsafe number of calls. Other than this, as would be expected, optimisation provides the greatest revenue gains.

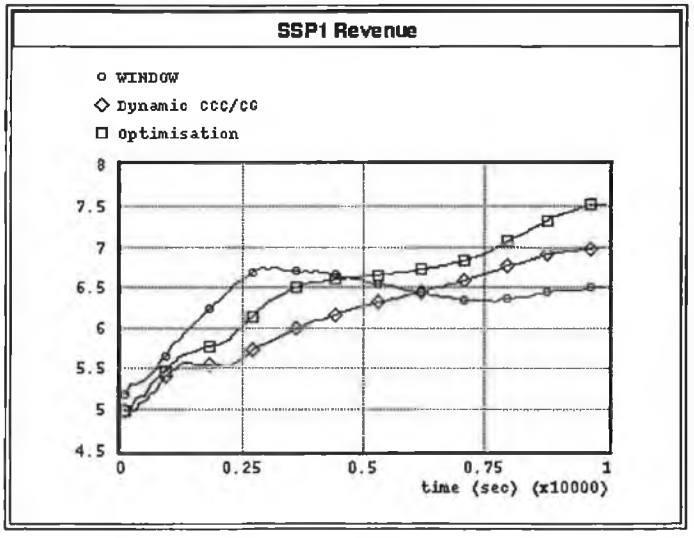


Fig. 6.27: SSP1 revenue for general overload

To summarise the results for general overload, both optimisation and dynamic CCC/CG protect all elements at all times, whereas Window fails to protect the SSP during SCP overload and overprotects it during SSP overload. Also, optimisation provides the best efficiency levels when the SCP is more overloaded than the SSP, while dynamic CCC/CG provides premium performance when the SSP overload exceeds that of the SCP.

6.3.5 Scenario 5: Overload due to Bursty Traffic

For this scenario, a 1000 second burst of televoting calls is applied to all SSPs every 2000 seconds. This causes simultaneous overload of the SCP and all SSPs, as shown in Figure 6.28, where the offered traffic to each physical element is expressed in terms of the capacity of that element.

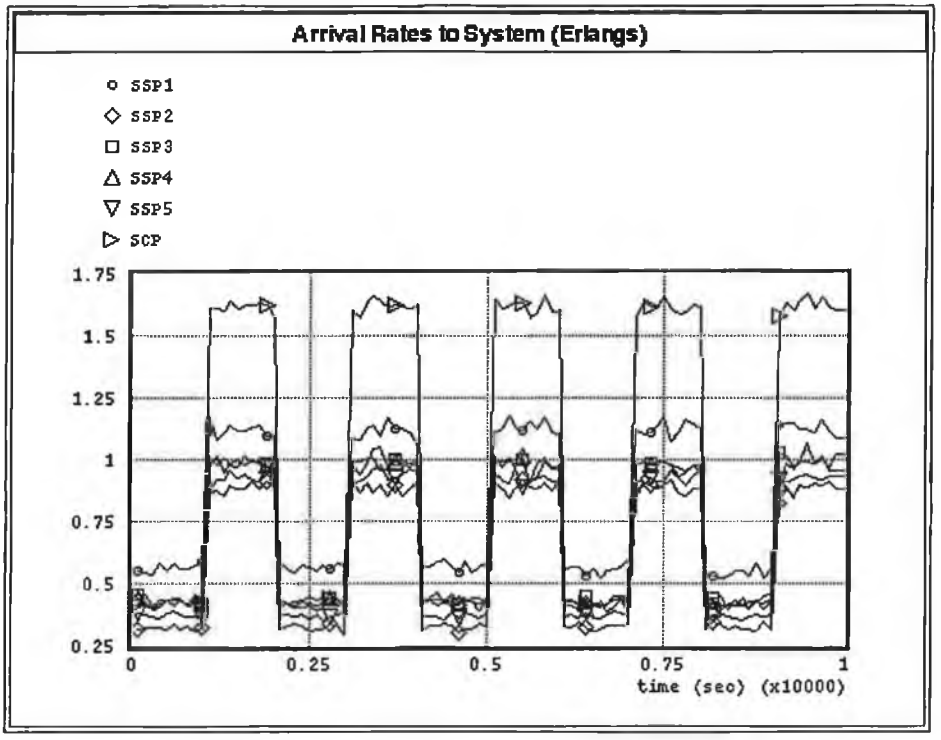


Fig. 6.28: Arrival rates for all IN physical elements

The resultant SCP load for each congestion control strategy is shown in Figure 6.29. Note that Window provides the best result here – it responds immediately both to the onset and termination of each traffic burst. For the other two strategies, the monitoring delay before congestion is detected results in the SCP load climbing to 1.0 Erlang and the SCP queue length growing to approximately 3000. When detection occurs, both dynamic CCC/CG and optimisation put the correct SCP throttles in place at Q2 of each SSP to alleviate the overload situation, but the load of the SCP does not descend to 0.8 for a few monitoring intervals, as the excess of calls which were queued at the SCP during the original monitoring delay must first be processed. In a similar manner, there is a delay of a maximum of one monitoring interval before the cessation of overload is detected by either dynamic CCC/CG or optimisation, during which an excess of calls are rejected. However, in this instance both strategies recover very quickly (as there is no SCP queue build-up) and put the correct controls in place immediately on detection of the change in the overload situation.

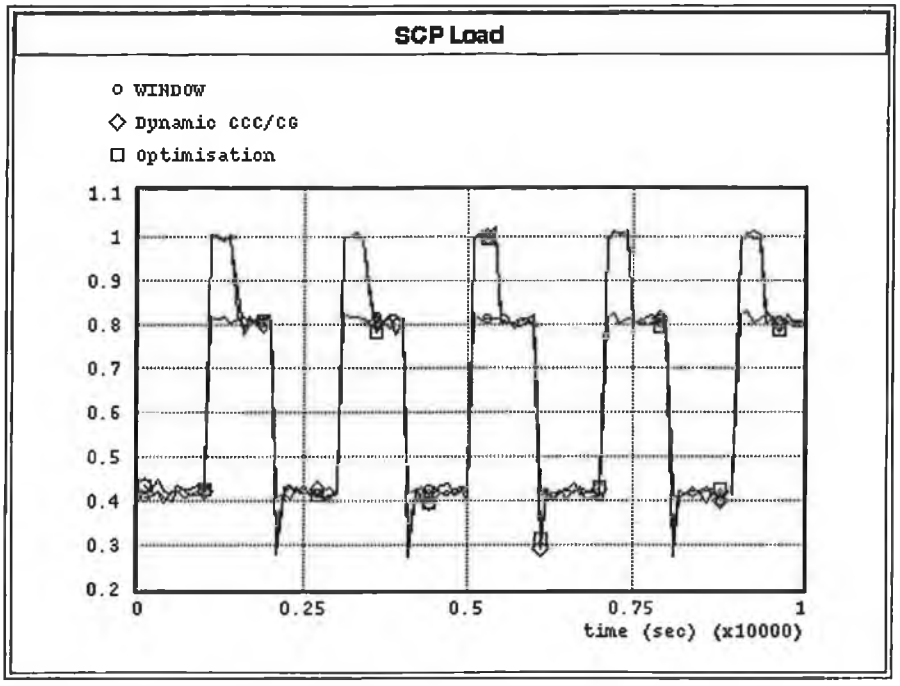


Fig. 6.29: SCP load for bursty overload

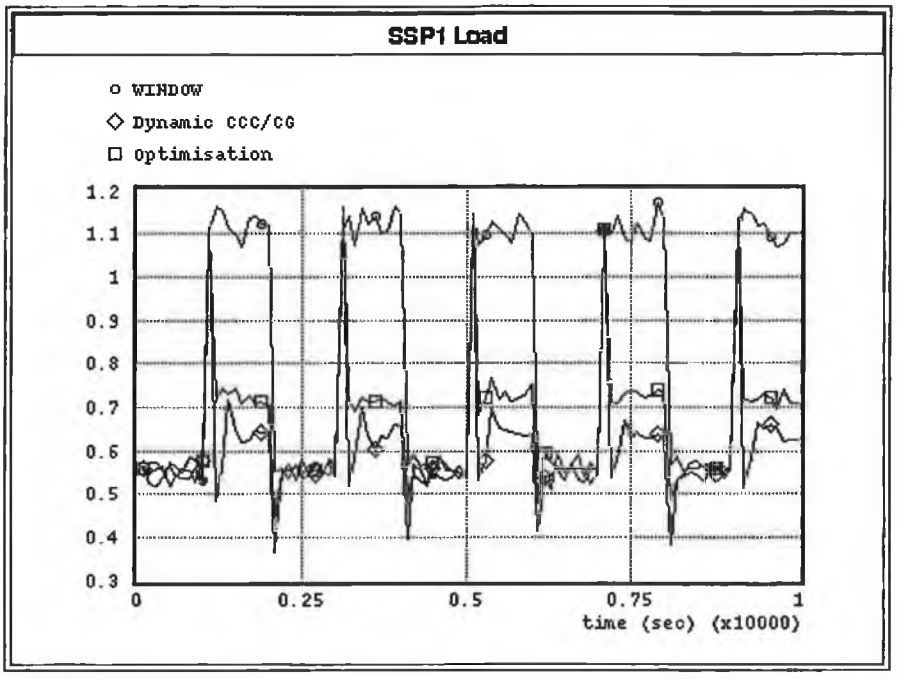


Fig. 6.30: SSP1 load for bursty overload

The behaviour of each strategy at the SSPs is very different, as may be seen in Figure 6.30. The Window-based strategy, as per usual, fails to protect SSP Q2 from overload for two reasons – firstly, as it fails to place emphasis on televoting load requirements, it does not calculate the SSP overload level correctly and puts insufficient throttles in place at Q1 and secondly, because it does not reject any traffic at Q2. Therefore, the load of Q2 for this strategy remains above 1.0 Erlang and its length rises to approximately 2000. On cessation of the traffic burst, Window again detects

the alleviation of the overload situation immediately, but experiences a small delay in reducing the SSP load, as it must complete processing of all calls that built up in the buffer of Q2 during the overload.

The other strategies, after the usual monitoring delay, detect overload and respond accordingly. However, unlike previous scenarios, the responses of dynamic CCC/CG and optimisation are not similar when both SCP and SSPs overload simultaneously. Optimisation, as a global strategy, takes the state of both the SCP and SSP into account before putting throttles in place at SSP Q2. It realises, therefore, that putting throttles in place to alleviate the SCP overload will also be sufficient to alleviate the SSP overload and therefore the SSP load level converges very quickly and only televoting calls are rejected in Q2 (as shown in Figure 6.31). Dynamic CCC/CG, on the other hand, seeks to protect each element independently. Therefore, the SCP detection algorithm puts controls in place at SSP Q2 to protect the SCP, while the SSP detection algorithm (without referring to the SCP throttles being put in place simultaneously) puts a throttle at Q1. The resulting conflict between controls means that an excess of calls are rejected during the following interval and oscillations occur in the load for the duration of the burst, while both SCP and SSP controls attempt to regulate the input traffic. On cessation of the traffic burst, both dynamic CCC/CG and optimisation reject calls unnecessarily for the remaining duration of that monitoring interval, after which time, both respond correctly by removing all controls.

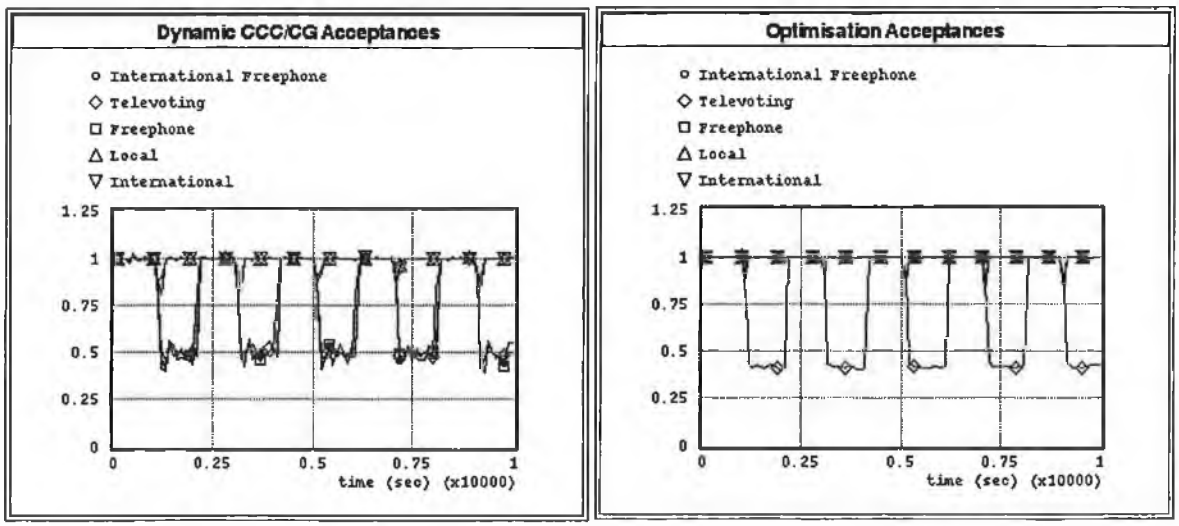


Fig. 6.31: SSP1 acceptances for bursty overload

Regarding call acceptances, optimisation accepts 5.1% more of the offered calls than does dynamic CCC/CG and 5.0% more calls than Window. There are two reasons for this – optimisation does not reject any non-IN calls and also, by rejecting only televoting, allows more low load-requiring IN calls to be processed at the SCP. Partially due to the greater number of acceptances, but also due to the fact that the types of calls accepted by optimisation are worth more financially, this strategy provides greater revenue gains than both other strategies.

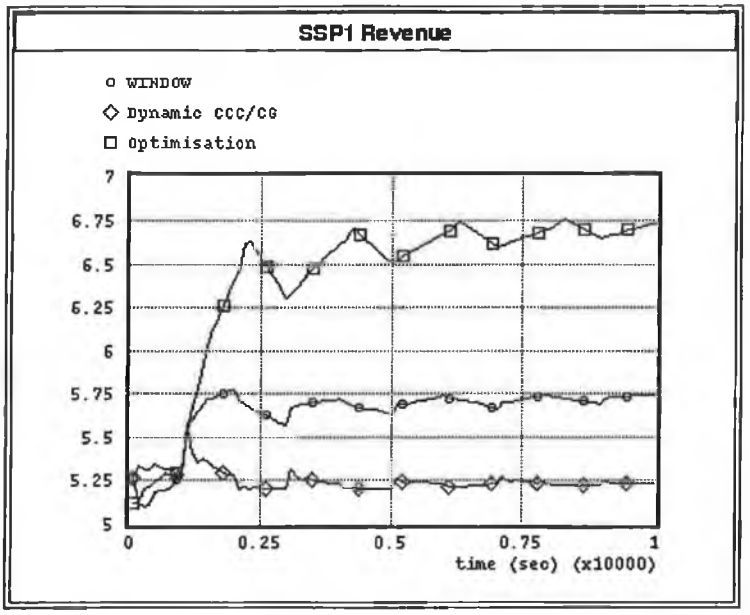


Fig. 6.32: SSP1 revenue for bursty overload

A final comparison that may be made between the strategies for this scenario is between service delays, as shown in Figure 6.33. Note that all IN service delays are excessive, as are non-IN delays for Window. For both dynamic CCC/CG and optimisation, these delays are as a result of delays at the SCP when the queue length there is large. Window, on the other hand, causes all services (IN and non-IN) to experience great delays at Q2. The only acceptable delay results are those experienced by non-IN calls subject to dynamic CCC/CG and optimisation.

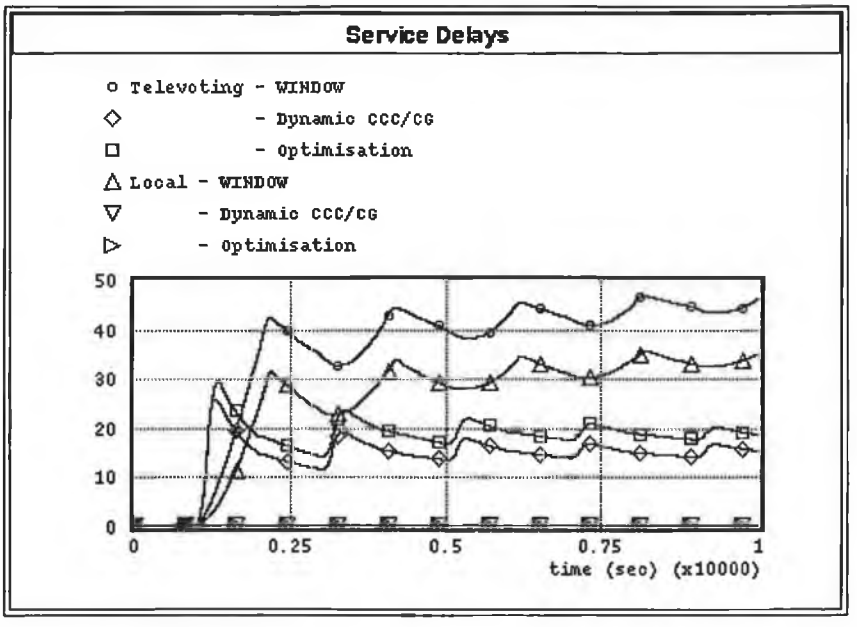


Fig. 6.33: SSP1 service delays for bursty overload

To summarise the results of this scenario, none of the strategies provide acceptable results in all areas. Window seems to provide the best overall results in that it reacts immediately to the onset of a burst and therefore protects the SCP. However, it fails to protect the SSP from overload. Both

dynamic CCC/CG and optimisation fail to protect the SCP from the onset of a burst, but protect the SSP adequately (optimisation providing better results). Therefore all strategies allow overload to occur at some point in the network, resulting in unacceptable post-dialling delays.

The only possible conclusion, therefore, is that any strategy containing a reactive component (i.e. a method or algorithm which reacts to an overload which is detected based on monitoring the variation of some value over an interval) cannot protect against an instantaneous dramatic increase in input traffic and that the only way to ensure against an overload of this type is to provide some sort of active strategy to act as an instantaneous cut-off point at the input to each physical element in the network. With just such a strategy in place to protect against the unlikely event of instantaneous overload, other reactive strategies and algorithms may then be used to intelligently protect the switch for all other input traffic scenarios.

6.4 Summary & Conclusions

The salient features of each of the strategies are outlined in Table 6.1 below, where a √ denotes acceptable behaviour and (*) denotes best behaviour for each category.

Category	*Classic CCC/CG*	*Window*	*Dynamic CCC/CG*	*Optimisation*
Relative processing requirements	3	9	1	60
Effectiveness of SCP protection for all traffic mixes and loads		√	√(*)	√(*)
Effectiveness of SSP protection for all traffic mixes and loads			√	√(*)
Throughput			√ (*) for SSP overload	√ (*) for SCP overload
Revenue gain			√	√(*)
Response to instantaneous overload		(*)		
Speed of convergence		√(*)	√	√(*)
Scalability			√	√
Flexibility				√(*)
Fairness			Subscriber	Subscriber Service

Table 6.1: Summary of Features for IN Congestion Control Strategies

To summarise these results, classic CCC/CG was found to have the worst overall response because it is based on the use of fixed call count and CG parameters and there are a number of issues associated with this use of fixed parameters. Firstly, call count and CG parameters are dependent on the size of resource at which they are located and must therefore be evaluated each time the algorithm is put in place at a different size resource – this is a non-trivial task. Secondly, it is impossible to define optimal call count parameters which work well over all possible input traffic mixes, as defining the parameters of necessity pre-supposes either that all calls require the same amount of processing or that the traffic mix does not vary, which is never the case. In other words, the problems with classic CCC/CG are basically an issue of scalability – the algorithm does not scale, either in terms of resource size or traffic mix.

A similar issue of scalability applies to the Window-based strategy. The Window timer duration and SSP CCC/CG algorithm are both based on the use of fixed parameters and cannot therefore react correctly for all traffic mix variations. This is proved in the section 6.3, where Window tends to overprotect the SCP when the bulk of applied traffic has high SCP processing requirements (and therefore greater average response delays than the Window timer duration) and underprotect it when the overload is caused by calls with low processing requirements and mean delays shorter than the Window timer duration. The effects of the SSP CCC/CG part of the strategy are even more noticeable – this algorithm either completely fails to protect the SSP or overprotects it considerably. In fact, the only advantage of using a Window-based strategy is that, due to its active nature, it provides the remote physical element that it is protecting with resistance to instantaneous dramatic increases in load levels. However, as this strategy is quite processor-hungry (requiring approximately three times more processing resource than static CCC/CG, as described in Chapter 4, section 4.3.4.4), a simple cut-off mechanism on the input buffer of each physical element would provide the same benefit with fewer processor requirements and could also be used in conjunction with reactive strategies, which provide consistently better results for all other traffic variations.

The dynamic CCC/CG algorithm is scalable. The only parameters that need to be set to target it to a particular resource are the capacity of that resource and the relative load requirements of each service type using it. In terms of monitoring overheads, it requires that the arrival rates for all calls types at the SCP must be monitored separately. Given this information, it can predict the overall impact of new arrivals on the resource load and calculate the appropriate throttles accordingly. As such, dynamic CCC/CG has very good performance during all overload levels and all traffic mixes. In fact, it provides equally good protection for all physical elements as the more complex optimisation strategy and is far superior to the classic CCC/CG and Window strategies. In fact, dynamic CCC/CG is more efficient than all other strategies in two ways. Firstly, the algorithms themselves actually have lower processing overheads than classic CCC/CG (by a factor of three)

and optimisation (by a factor of sixty (for the LP_SOLVE software)) and secondly, as dynamic CCC/CG rejects calls efficiently at Q1 during SSP overload (as opposed to optimisation, which does not reject any calls until Q2), it provides greater SSP (and therefore IN) throughput for this scenario. Note also that dynamic CCC/CG exhibits subscriber fairness in that the gap values associated with the CG throttles in the SSPs are evaluated from the percent thinning coefficients sent to them by the SCP – this combines the subscriber fairness of PT with the efficiency of CG (as described in Chapter 4). The only desirable characteristic not demonstrated by dynamic CCC/CG is flexibility – the algorithm does not easily lend itself to being extended to include selective throttling of service types based on e.g. priorities or focussed overload.

The optimisation-based algorithm also provides excellent results. It has all the advantages of dynamic CCC/CG in terms of scalability (for both resource targeting and handling of variations in traffic mix) and subscriber fairness. In terms of monitoring overheads, it requires that the arrival rates for all calls types at both the SCP and SSPs must be monitored separately. However, it does have a number of other advantages not associated with dynamic CCC/CG. The strategy is innately flexible, and can be extended to encompass other requirements by either re-specification of call weights or by the inclusion of other constraints in the maximisation algorithm. Service fairness, as well as subscriber fairness, is always preserved (within the bounds of the priority system). Priorities allocated to service types are always honoured, even during congestion. The interoperable nature of the SCP and SSP algorithms in the strategy also ensures premium IN performance during SCP overload, and revenue in the IN is maximised at all times, without compromising fairness or user delays. All these advantages, however, do not come without a price. The first negative aspect of the optimisation strategy is that its processing overheads are so much greater than for dynamic CCC/CG. However, two points may be raised with regard to this:

- if the optimisation overheads are related to the processing requirements of service requests on the SCP, using the optimisation strategy equates to the loss of only one freephone call per monitoring interval and the resultant gains in IN throughputs and revenue achieved by using the optimisation strategy are sufficiently high during SCP overload to render this overhead negligible,
- the LP_SOLVE optimisation software used in the simulations is a two-phase simplex algorithm designed to optimise much more complex LPPs than the single-phase optimisation strategy investigated here. Therefore, if the optimisation-based congestion control software were to be streamlined (as would be required if it were to be used in a real system), processing overheads would be likely to be considerably lower.

The second negative aspect of the optimisation strategy is more considerable and relates to the operation of optimisation during SSP overload. During this scenario, the processing overheads

associated with accepting all calls at Q1 (so that they may be differentiated at Q2) are considerable, and so this aspect of the optimisation strategy is not satisfactory. The solution would seem to be to merge the SSP optimisation algorithm with aspects of the SSP dynamic CCC/CG algorithm to produce a hybrid that takes the current state of both Q2 and the SCP into account when devising the global throttles to be put in place on all traffic at Q1. In this manner, the operation of optimisation would then be either equal or superior to all other strategies at all times.

To conclude, the effectiveness of both the optimisation and dynamic CCC/CG strategies are equivalent and far superior to either Window or classic CCC/CG. Both strategies also exhibit scalability and subscriber fairness (unlike Window or classic CCC/CG) and dynamic CCC/CG is even more efficient than classic CCC/CG, in terms of requiring lower processing overheads to execute. Optimisation, on the other hand, provides more flexibility, service fairness and better revenue than both classic and dynamic CCC/CG, but at the expense of significantly greater processing overheads.

Chapter 7

Conclusions & Recommendations

7.1 Conclusions of this Work

The primary conclusion of this work relates to the types of congestion control strategies that should be used for IN protection. Static strategies based on the use of tables of fixed parameters (e.g. CCC, LMC, CG and Window) should not be used, as they are incapable of protecting the IN under varying loads and traffic mixes and therefore fail to meet even the basic requirements on a congestion control strategy – this was proved in Chapters 4 and 6 of this thesis. Instead, the application of scalable dynamic strategies is recommended, as they have a number of advantages, including:

- Their scalability makes them extremely easy and fast to target to a particular resource,
- They respond correctly to any variations in traffic load,
- They can handle any variations in traffic mix – i.e. they can take into account the fact that different request types have different processing requirements at different resources in the network and respond accordingly.

In other words, the use of dynamic detection methods in conjunction with dynamic throttles means that, not only is the system scalable, but also the overload controls put in place are, at any time, for any traffic mix, exactly appropriate for the level of overload. Two such strategies are presented in this work – the revenue optimisation strategy described in Chapter 5 and the dynamic CCC/CG strategy introduced in Chapter 6. Both of these strategies provided far superior results, in terms of both effectiveness and efficiency, than any of the strategies most commonly used in industry today.

The optimisation strategy, as well as being dynamic, has the added advantage of being a global IN strategy, in that it takes the state of both the SCP and SSP into account when determining the overload level in the network, and puts the appropriate controls in place to protect both PEs. This means that when the SCP and SSPs of an IN are suffering from congestion, while other strategies attempt to protect each PE independently and as a result reject too many calls overall, optimisation ensures optimum IN performance at all times. Optimisation is also very flexible and can selectively throttle different call types based on, for example, their relative importance (as defined by the IN service provider), their revenue, their applied load and their different load requirements at both the SCP and SSPs. However, this extra level of intelligence does not come without a price – optimisation has considerably more processing overheads than does dynamic CCC/CG, both in terms of the footprint of the algorithm and the fact that all calls must be processed in the SSP to the point where differentiation between call types, and therefore selective throttling, is possible.

We therefore recommend that in Intelligent Networks where SCP overload is more usual than SSP overload or where priorities, service fairness or revenue are an issue, optimisation should be used as the benefits of its use here far outweigh its greater processing overheads, while in networks where all calls are to be treated equally, dynamic CCC/CG should be the preferred strategy.

A number of other conclusions may also be presented, based on observations made during the course of this research. The first of these is that it is absolutely critical that when a model is developed to investigate congestion control, it should reflect the real network architecture, functionality and its applied traffic as much as possible, in order to ensure that the research carried out on it is valid and the results dependable. Chapter 2 described a significant amount of research into the applicability of parameter-based congestion control algorithms in the IN arena. The results of this research were generally positive, in that most of the strategies were perceived to succeed at protecting the IN from overload. However, there was a fundamental flaw in much of this research – most of the models used were very much over-simplified and in general, the behaviour of the strategies was only investigated under an applied load of one traffic type (or when more than one type was used, it was generally assumed that all types of requests had the same load requirements at the SCP). As a result, the fact that parameter-based strategies are incapable of dealing with different traffic types with different load requirements was not recognised. The model presented in Chapter 4 reflected the architecture of the IN in enough detail (as well as the information flows between PEs for a number of different services) that the limitations of these strategies became immediately apparent. Therefore, it is highly recommended that, to ensure the validity of a body of research, a sufficiently detailed model of the target network be developed – this has a greater cost, in terms of development time, but ensures that the results acquired will be valid.

Another conclusion of this work relates to the two most commonly used throttles in IN congestion control – namely, percent thinning and call gapping. It was verified in [Berger91] and Chapter 4 (section 4.3.3) that while PT has the advantage of exhibiting both subscriber fairness and scalability, CG exhibits robustness and a faster response to the onset of congestion. A logical conclusion of this is that a combination of the two would combine the advantages of each to produce a flexible and scalable throttle with both subscriber fairness and robustness. In this way, the output of any SCP detection algorithm should be a PT coefficient (to ensure scalability and subscriber fairness) and this should be translated in each SSP into a gap interval, which will ensure robustness. This throttle would also remove the principle disadvantage associated with CG, i.e. its parameter-driven nature – instead of using fixed parameters, an appropriate gap interval is calculated based on a PT coefficient. The use of the PT/CG throttle algorithm described in Chapter 6, section 6.2 as part of the dynamic CCC/CG strategy is therefore recommended.

The final conclusion presented here relates to the behaviour of all strategies under bursty overload – only Window and QLC (i.e. strategies with no monitoring intervals and a tight control loop) are capable of responding quickly enough to the onset of bursty traffic. However, neither strategy behaves consistently enough under other traffic loads to deserve recommendation (they tend to react to overload even when no overload exists) – to make intelligent decisions about how to manage an overload, a monitoring period is required to allow the congestion control strategy to base its controls on the mean state of the system. It is therefore recommended that performance management of any system should be carried out at two levels. At the lower level, all physical entities (or nodes) in a network should have a simple active strategy of some sort at their input that ensures against instantaneous overload, so that each PE is responsible for crisis management. However, this mechanism should only reject enough requests to ensure the survival of its PE. This is so that a global congestion control strategy (the higher level of the performance management strategy) can make decisions, based on observation of the mean state of the network, about how to throttle traffic intelligently in different PEs in order to acquire the best possible overall network performance.

7.2 Recommendations for Future Work

The current behaviour of the optimisation strategy is not ideal – too much SSP processing resource needs to be applied to progress requests to the point where selective throttling is possible. As suggested in the conclusions of Chapter 6 (section 6.4), it might be useful to investigate how to merge the SSP optimisation algorithm with the SSP part of the dynamic CCC/CG strategy so as to acquire a global IN congestion control strategy which combines selective throttling of all calls at SSP Q2 with some global throttling of calls at SSP Q1. In this way, some of the advantages of selective throttling may be retained, while maximising the throughput of the network at all times.

Further work also needs to be carried out based on the enhancement of the IN architecture in CS-2 [Q.1221]. Specifically, there is much potential for using SCP/SCP interworking as a flow control mechanism in overload situations, but it would need to be managed intelligently. It might therefore be interesting to investigate whether it is possible to extend the optimisation algorithm to encompass the management of multiple SCPs in a single IN domain, so that all PEs – SCPs and SSPs alike – co-operate to provide optimum Intelligent Network performance.

Appendix A

References

[Akyildiz90] I.F. Akyildiz, R. Shonkwiler, "Simulated Annealing for Throughput Optimisation in Communication Networks with Window Flow Control", IEEE Conference on Communications, 1990.

[Angelin95] L. Angelin, A. Arvidsson, "A Congestion Control Mechanism for Signaling Networks based on Network Delays", Proceedings of the 12th Nordic Teletraffic Seminar, Helsinki, 1995.

[Arvidsson96] A. Arvidsson, S. Pettersson, L. Angelin, "Congestion Control in Intelligent Networks for Real Time Performance and Profit Optimisation", Proceedings of ITC Specialist Seminar, Lund 1996.

[ATM99] http://www.atmforum.com.

[Bellcore92] Bellcore, "Advanced Intelligent Network (AIN) 0.1 Switching Systems Generic Requirements", Technical Reference TR-NWT-001284, Issue 1, August 1992.

[Berger91a] A. Berger, "Determination of Load-Service Curves for Distributed Switching Systems: Probabilistic Analysis of Overload-Control Schemes", ITC-13, Copenhagen, 1991.

[Berger91b] A. Berger, "Comparison of Call Gapping and Percent Blocking for Overload Control in Distributed Switching Systems and Telecommunications Networks", IEEE Trans. Commun., 39, pp 407-414, 1991.

[Berkelaar95] The Linear Programming Toolkit, LP_SOLVE version 2.0, developed by Michel Berkelaar, Eindhoven University of Technology and Jeroen Dirks, Delft University of Technology, 21.2.95.

[Bolotin94] V.A. Bolotin, "Telephone Circuit Holding Time Distributions", ITC-14, Juan-les-Pins, 1994.

[Burkard83] L. Burkard et al, "Customer Behaviour and Unexpected Dial Tone Delay", Proceedings of ITC-10, Montreal, 1983.

[CORBA99] "CORBA/IIOP 2.3 Specification", available from http://www.omg.org/library/c2indx.html

[Daisenberger85] G. Daisenberger, J. Oehlerich, G. Wegmann, "STATOR - STATistical Overload Regulation - and TAIL - Time Account Input Limitation - Two concepts for overload regulation in SPC switching systems", ITC-11, Kyoto, 1985.

[Daisenberger89] G. Daisenberger, J. Oehlerich, G. Wegmann, "Two concepts for overload regulation in SPC switching systems: STATOR and TAIL", Telecommunication Journal, Volume 56, pp 306-313, 1989.

[Doshi91] B. Doshi, H. Heffes, "Overload Performance of an Adaptive, Buffer-Window Allocation Scheme for a Class of High Speed Networks", ITC-13, Copenhagen, 1991.

[Dziong89] Z. Dziong, M. Pioro, U. Korner, T. Wickberg, "On Adaptive Call Routing Strategies in Circuit Switched Networks - Maximum Revenue Approach", ITC-12, Turin, 1989.

[EricssonCS1+] http://www.ericsson.com/NI/product/platforms/scpt.html.

[E.721] ITU-T Recommendation E.721, " Network Grade of Service Parameters and Target Values for Circuit-Switched Services in the Evolving ISDN."

[E.723] ITU-T Recommendation E.723, "Grade of Service Parameters for Signalling System Number 7 Networks."

[Galletti92] M. Galletti, F. Grossini, "Performance Simulation of Congestion Control Mechanisms for Intelligent Networks", Proceedings of 1992 International Zurich Seminar on Digital Communications, Intelligent Networks and their Applications, Zurich, 1992.

[Gelenbe] E. Gelenbe, G. Pujolle, *Introduction to Queueing Networks*, John Wiley & Sons, New York, Chichester, Brisbane, Toronto, Singapore, 1987.

[Greenberg] M.R. Greenberg, *Applied Linear Programming for the Socioeconomic and Environmental Sciences*, Academic Press, New York, San Francisco, London, 1987.

[Gulyani93] M. Gulyani, "Simulation and Performance Analysis of an Telecommunication System Based on Advanced Intelligent Network Architecture", Masters Thesis, Dublin City University, Ireland, 1993.

[Hac98] A. Hac, L. Gao, "Congestion Control in Intelligent Network", IEEE Internation Performance, Computing and Communications Conference (IPCCC'98), Phoenix, 1998.

[Harrison] P.G. Harrison, N.M. Patel, *Performance Modelling of Communication Networks and Computer Architectures*, International Computer Science Series, Addison-Wesley Publishing Company, 1993.

[Hebuterne90] G. Hebuterne, L. Romoeuf, R. Kung. "Load Regulation Schemes for the Intelligent Network", XIII International Switching Symposium, Stockholm, May 1990.

[Hubig94] W. Hubig, D. Weber, "Overload Control in ISDN PABXs", ITC-14, Juan-les-Pins, 1994.

[Hoang90] B. Hoang, "Service Completion Time for Advanced Intelligent Network Services", IEEE Conference on Communications, 1990.

[ITU_IN] Intelligent Network Standards, Q.12XX series recommendations, International Telecommunications Union – Telecommunications Standardisation, 1993-1994.

[ITU_ISDN] Integrated Services Digital Network Standards, I-series recommendations, International Telecommunications Union – Telecommunications Standardisation, 1988-present.

[ITU_TMN] "Overview of TMN Recommendations", Document M.3000, ITU-T, October 1994.

[Kallenberg89] P.J.M. Kallenberg, "Load Estimation for Overload Control", ITC-12, Turin, 1989.

[Kant95] K. Kant, "Performance of Internal Overload Controls in Large Switches", IEEE, pp 228-237, 1995.

[Kawamura96] H. Kawamura, E. Sano, "A Congestion Control System for an Advanced Intelligent Network", Proceedings of IEEE Network Operations and Management Symposium (NOMS), 1996.

[Kihl95] Maria Kihl, "Overload Control in Intelligent Networks", Lund University, Sweden, 1995.

[Kihl97] M. Kihl, M. Rumsewicz, "Analysis of overload control strategies in combined SSP-SCPs in the Intelligent Network", ITC-15, 1997.

[Kleinrock] L. Kleinrock, *Queueing Systems Volume I: Theory*, John Wiley & Sons, New York, Chichester, Brisbane, Toronto, 1975.

[Korner91] Ulf Korner, "Overload Control of SPC Systems", Proceedings of ITC-13, pp 105-114, Copenhagen, 1991.

[Korner94] Ulf Korner, C. Nyberg, B. Wallstrom, "The Impact of New Services and New Control Architectures on Overload Control", ITC-14, Juan-les-Pins, 1994.

[Kwiatkowski94a] M. Kwiatkowski, "Queue Length Congestion Control at an SCP", ATNAC, Melbourne, 1994.

[Kwiatkowski94b] M. Kwiatkowski, B. Northcote, "Calculating Mean Delays in Intelligent Networks Under Overload", ATNAC, Melbourne, 1994.

[Langlois91] F. Langlois, J. Regnier, "Dynamic Congestion Control in Circuit-Switched Telecommunications Networks", ITC-13, Copenhagen, 1991.

[Lee97] Y. Lee, J.S. Song, "Overload Control of SCP in Advanced Intelligent Network with Fairness and Priority", Proceedings of the 6th International Conference on Computer Communications & Networks, Las Vegas, 1997.

[Leever93] P.J.E. Leever, G.S. Vermeer, R.A.J. Reijmerink, L.J.N. Franken and B.R. Haverkort, "Performance Evaluation of Intelligent Network Services", Tenth UK Teletraffic Symposium, Performance Engineering in Telecommunications Networks, April 1993.

[Lindberg88] P. Lindberg, K. Nivert, B. Sagerholm, "Trunk Reservation and Grade of Service Issues in Circuit Switched Integrated Networks", ITC-12, Turin, 1988.

[Luan89] D.T.D. Luan, D.M. Lucantoni, "Throughput Analysis of an Adaptive Window-Based Flow Control Subject to Bandwidth Management", ITC-12, Turin, 1989.

[MacDonald94] D.M. MacDonald, S. Archambauld, "Using Customer Expectation in Planning the Intelligent Network", ITC-14, Juan-les-Pins, 1994.

[Manfield91] D. Manfield, B. Denis, K. Basu, G. Rouleau, "Overload Control in a Hierarchical Switching System", Proceedings of ITC-13, pp 894-900, Copenhagen, 1991.

[Milito91] R.A. Milito, Y. Levy, Y. Arian, "Dynamic algorithms for distributed queues with abandonment", ITC-13, Copenhagen, 1991.

[Newcombe94] A. Newcombe, D.D. Botvich, F. Lodge, T. Curran, "A decision support system for assurance of quality of service in intelligent network service provisioning", Proceedings of IS&N '94, Aachen, September 1994.

[Nyberg92] C. Nyberg, "On Overload Control in Telecommunication Systems", Technical Report-111, Department of Communication Systems, University of Lund, Sweden, 1992.

[Nyberg94] H. Nyberg, B. Olin, "On Load Control of an SCP in the Intelligent Network", ATNAC, Melbourne, 1994.

[Nyberg95a] C. Nyberg, M. Kihl, "Overload Control in Intelligent Networks: an Approach using Modified PID Controllers", University of Lund, Sweden, 1995.

[Nyberg95b] C. Nyberg, M. Kihl, U. Ahlfors, U. Korner, "The Impact of Retrials on Overload Control", Internal document, Lund, 1995.

[Pham91] X.H. Pham, "Control loop for traffic management of networks under focussed overloads", ITC-13, Copenhagen, 1991.

[Pham92] 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.

[Q1202] "Intelligent Network Service Plane Architecture", Document Q.1202, ITU-T, April 1993.

[Q1213] "Global Functional Plane for Intelligent Network CS-1", Document Q.1213, ITU-T, January 1994.

[Q1214] "Distributed Functional Plane for Intelligent Network CS-1", Document Q.1214, ITU-T, February 1994.

[Q1215] "Physical Plane for Intelligent Network CS-1", Document Q.1215, ITU-T, November 1993.

[Q1218] "Interface Recommendations for Intelligent Network CS-1", Document Q.1218, ITU-T, January 1994.

[Q1221] "Introduction to Intelligent Network Capability Set 2", Document Q.1221, ITU-T, September 1997.

[Roberts79] J. Roberts, "Recent Observations of Subscriber Behaviour", Proceedings of ITC-9, Spain, 1979.

[Rajaratnam96] M. Rajaratnam, F. Takawira, "Modelling multiple Traffic Streams Subject to Trunk Reservation in Circuit-Switched Networks", GLOBECOM '96,

[Rumsewicz95] M. Rumsewicz, "On the real-time determination and control of mass call-ins in Intelligent Networks", Software Engineering Research Centre Technical Report SERC-0003, the Royal Melbourne Institute of Technology, October 1995.

[Rumsewicz96] M. Rumsewicz, "A simple and effective algorithm for the protection of services during SCP overload", Proceedings of the 4th International Conference on Telecommunications Systems, 1996.

[Sabourin91] T. Sabourin, G. Fiche, M. Ligeour, "Overload Control in a Distributed System", Proceedings of ITC-13, pp 421-427, Copenhagen, 1991.

[Seraj85] Jila Seraj, "An analysis of processor load control in SPC systems", Proceedings of ITC-11, pp 767-773, Kyoto, 1985.

[Smith95] D.E. Smith, "Ensuring Robust Call Throughput and Fairness for SCP Overload Controls", IEEE/ACM Transactions on Networking, Vol. 3, No. 5, pp 538-548, October 1995.

[Swenson96] E. Swenson, "ITU-T Intelligent Network Capability Set 2 Recommendations", ITC Mini-Seminar on Engineering and Congestion Control in Intelligent Networks, Australia, 1996.

[TINA97] "Service Architecture Version 5.0", TINA Consortium, June 1997, available from http://www.tinac.com.

[Tsolas92] N. Tsolas, G. Abdo, R. Bottheim, "Performance and Overload Considerations when Introducing IN into an Existing Network", International Zurich Seminar on Digital Communications, Zurich 1992.

[Turner91] P.M.D. Turner, P.B. Key, "A New Call Gapping Algorithm for Network Traffic Management", ITC-13, Copenhagen, 1991.

[Villen85] M. Villen-Altamirano, G. Morales-Andres, L. Bermejo-Saez, "An Overload Control Strategy for Distributed Control Systems", Proceedings of ITC-11, pp 835-841, Kyoto, 1985.

[Wallstrom91] B. Wallstrom, C. Nyberg, "Transient Model of Overload Control and Priority Service in SPC Systems", Proceedings of ITC-13, pp 429-434, Copenhagen, 1991.

[Yan94] J. Yan, D.M. MacDonald, "Teletraffic Performance in Intelligent Network Services", ITC-14, Juan-les-Pins, 1994.

[Zepf91] J. Zepf, G. Willmann, "Transient Analysis of Congestion and Flow Control Mechanisms in Common Channel Signalling Networks", ITC-13, Copenhagen, 1991.

Appendix B

References Associated with this Research

[Lodge94] F. Lodge, T. Curran, M. Gulyani, A. Newcombe, "Intelligent Network Congestion Control Strategies and their Impact on User-Level Quality of Service", Proceedings of Australian Telecommunication Networks & Applications Conference, pp 627-632, Melbourne, December 1994.

[Lodge96] F. Lodge, T. Curran, "A Congestion Control Strategy for Combined IN and non-IN Traffic Load at the Service Switching Point of an Intelligent Network ", Proceedings of Networks '96, Sydney, December 1996.

[Lodge97] Fiona Lodge, Dmitri Botvich, Thomas Curran, "A fair algorithm for throttling combined IN and non-IN traffic at the SSP of the Intelligent Network", Proceedings of IEE Teletraffic Symposium, Manchester, March 1997.

[Lodge98a] Fiona Lodge, Dmitri Botvich, Thomas Curran, "A Fair Intelligent Network Congestion Control Strategy Based on Revenue Optimisation", Proceedings of IS&N'98, Antwerp, May 1998.

[Lodge98b] F. Lodge, B. Jennings, T. Curran, "A Strategy For The Resolution of Intelligent Network (IN) and Signalling System No. 7 (SS7) Congestion Control Conflicts" Proceedings of ICC '98, Atlanta, June 1998.

[Lodge99] Fiona Lodge, Dmitri. Botvich, Thomas Curran, "Using Revenue Optimisation for the Maximisation of Intelligent Network Performance", Proceedings of ITC-16, Edinburgh, June 1999.

[Newcombe94] A. Newcombe, D.D. Botvich, F. Lodge, T. Curran, "A decision support system for assurance of quality of service in intelligent network service provisioning", Proceedings of IS&N '94, Aachen, September 1994.

Appendix C

Glossary

ACG	Automatic Code Gapping
AIN	Advanced Intelligent Network
ATM	Asynchronous Transfer Mode
BCP	Basic Call Process
CCAF	Call Control Agent Function
CCC	Call Count Control
CCF	Call Control Function
cdf	Cumulative Distribution Function
CID	Call Instance Data
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
CS-x	IN Capability Set Number x
BCSM	Basic Call State Model
CG	Call Gapping
DFP	Distributed Functional Plane
DN	Destination Number
FDOC	Focussed Destination Overload Control
FE	Functional Entity
FEA	Functional Entity Action
FIFO	First In First Out
FSM	Finite State Machine
GFP	Global Functional Plane
IAF	Intelligent Access Function
IDL	Interface Description Language
IETF	Internet Engineering Task Force
IF	Information Flow
IIOP	Internet Inter-Orb Protocol
IN	Intelligent Network
INAP	Intelligent Network Application Part
INCM	Intelligent Network Conceptual Model
IP	Intelligent Peripheral
IP	Internet Protocol
ISDN	Integrated Services Digital Network
IT	Information Technology
ITU	International Telecommunications Union
LMC	Load Measure Control

LPP	Linear Programming Problem
O-BCSM	Originating Basic Call State Model
OMG	Object Management Group
ONP	Open Network Provisioning
OPNET	OPtimised Network Engineering Tools
pdf	Probability Density Function
PE	Physical Entity
PID	Proportional Integral Differential
PIN	Personal Identification Number
pmf	Probability Mass Function
POI	Point of Initiation
POR	Point of Return
PP	Physical Plane
PSTN	Public Switched Telephone Network
PT	Percent Thinning
QLC	Queue Length Control
Rev/Res	Revenue to Resource ratio
RTC	Response Time Control
RV	Random Variable
SCEF	Service Creation Environment Function
SCF	Service Control Function
SCP	Service Control Point
SDF	Service Data Function
SDP	Service Data Point
SIB	Service Independent Building Block
SLP	Service Logic Program
SLPI	Service Logic Program Instance
SMF	Service Management Function
SOC	SCP Overload Control
SOCC	SMS-Originated Code Control
SP	Service Plane
SPC	Stored Program Controlled
SRF	Service Resource Function
SS7	Signalling System No. 7
SSCP	Service Switching and Control Point
SSF	Service Switching Function

SSP	Service Switching Point
svc	Square of the Variation Coefficients
T-BCSM	Terminating Basic Call State Model
TCAP	Transaction Capabilities Application Part
TINA	Telecommunications Intelligent Networking Architecture
TMN	Telecommunication Management Networks
UI	User Interaction
VPN	Virtual Private Network