LUND UNIVERSITY

Overload control and performance evaluation in a Parlay/OSA environment

Andersson, Jens K

2004

Link to publication

Citation for published version (APA):
Andersson, J. K. (2004). *Overload control and performance evaluation in a Parlay/OSA environment*. [Licentiate Thesis, Department of Electrical and Information Technology]. Lund Institute of Technology.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Overload Control and Performance Evaluation in a Parlay/OSA Environment

Jens K Andersson

LUND UNIVERSITY

Department of Communication Systems

Lund Institute of Technology

ISSN 1101-3931
ISRN LUTEDX/TETS--1067--SE+100P
© Jens Andersson

Printed in Sweden
E-kop
Lund 2004

To Elisabeth

This thesis is submitted to Research Board FIME – Physics, Informatics, Mathematics and Electrical Engineering – at Lund Institute of Technology (LTH), Lund University, in partial fulfilment of the requirements for the degree of Licentiate in Engineering.

This work was partly funded by the Swedish agency for Innovative Systems (VINNOVA)

Contact Information:

Jens Andersson
Department of Communication Systems
Lund University
P.O. Box 118
SE-221 00 LUND
Sweden

Tel: +46 46 222 91 58
Fax: +46 46 14 58 23
E-mail: jens.andersson@telecom.lth.se
Web: http://www.telecom.lth.se/Personal/wiw/jensa

Abstract

To increase the pace of development and deployment of new services and applications in telecommunication networks, new service architectures have been proposed. Parlay/OSA is one of the proposals that has aroused most attention. By providing network functionality via Application Program Interfaces (APIs), Parlay/OSA facilitates creation of telecommunication services and applications for independent software developers. With Parlay/OSA there is no longer any requirement for knowledge and technical skills of telecommunications when creating new applications. A Parlay/OSA environment introduces gateways, which provide the applications with abstracted network functionality. A gateway translates the requests for abstracted functionality to telecommunication protocols and signalling. Parlay/OSA is a contract driven architecture where several constraints coexist. To fulfil the constraints and to protect the gateways from shutting down, overload control mechanisms are needed.

This thesis proposes and evaluates different overload control mechanisms in a Parlay/OSA environment. The main objectives of the overload control mechanisms are to protect the system and to maintain a high throughput. Important issues that are discussed are measurement strategies, overload control actions, and how different priorities and time constraints should be considered. Also, the proposed overload control mechanisms consider some guiding principles from the specifications of Parlay/OSA. In the scope of this thesis both distributed and single server systems are investigated.

Acknowledgements

First of all I would like to thank my two supervisors, Dr. Maria Kihl and Dr. Christian Nyberg for their contributions and support during my work with this thesis. I would also like to thank Prof. Ulf Körner for having me as a Ph.D. student at a very nice department. Thanks to all my colleagues at the department for making it such a nice place to work at. I am also very grateful to Appium AB and Daniel Söbirk. It has meant a lot for this thesis to have some industrial cooperation.

I would also like to thank my mother and father for always encouraging whatever I am doing. Another important person in my life is my brother and friend Tobias, who forced me to study hard in early days by letting me do his homework. Also thanks to all my friends, making my life so much richer. Finally and most I would like to thank Elisabeth who always supports me and makes my life outside work so enjoyable. I always have a great time with you.

Contents

Introduction

Recently, new service architectures have been developed to increase the pace of development of new services and applications for telecommunication networks. The main idea with the new proposals is to provide abstracted network functionality via Application Program Interfaces (APIs). Parlay and OSA are examples of service architectures that open up an underlying network to independent software developers via APIs. A Parlay/OSA architecture consists of several processing entities. The applications use so-called Application Servers (ASs) to communicate with a Parlay/OSA Gateway which processes the conversion from the APIs to the network functionality. If there are too many applications that want to make use of the gateway or AS at the same time these processing nodes must be protected from overload. It is too expensive to over dimension the capacity such that overload situations never occur. Neither it is possible to give any guarantees that overload will not occur with a certain capacity.

Overloaded nodes would correspond to long waiting times in the best case and servers that goes down in the worst case. Another important issue is that the new service architectures are based on contracts where different Service Providers (SPs) might have different amount of guaranteed accepted service calls per second, maximal delays etc. Therefore, overload control mechanisms are needed in order to maintain fairness and a sufficient Quality of Service (QoS).

This thesis investigates overload control for an AS and for a Parlay/OSA gateway. Overload control of a distributed system can be divided into two parts, namely load balancing and admission control. Load balancing algorithms are used to distribute the load such that all processing entities are exposed to about the same load. If it is not feasible to avoid overload with the load balancing algorithm, admission control algorithms are used. Admission control algorithms rejects some of the service requests during overloaded situations. The decisions of which requests that should be rejected can be based on different criteria. In this thesis for example algorithms for profit optimization is investigated. This means that the service requests generating most revenue are prioritized during overload situations.

The remainder of this thesis is organized as follows. In section 1 the background of the development of service architectures is presented. It is also discussed why overload control algorithms are needed. Discussions about services and applications are found in section 2. Especially so-called web services and future services are discussed. Section 3 describes classical admission control and load balancing algorithms. Advantages and drawbacks are also discussed. The papers included in this thesis are summarised in section 4 and further research is presented in section 5.

There are four research papers included in this thesis. The papers appear after section 5 and the papers included are the following in the order that follows:

Paper 1 - *Extended version of* Service Architectures for the Next Generation Networks: An Overview and Some Performance Aspects
Maria Kihl and Jens Andersson
Published in proceedings of the TINA Workshop, Malaysia, 2002

Paper 2 - Performance Analysis and Modelling of an OSA Gateway
Jens Andersson, Christian Nyberg and Maria Kihl
Published in proceedings of the Personal Wireless Communication, Venice, Italy, 2003

Paper 3 - Performance Analysis and Overload Control of an Open Service Access Architecture
Jens Andersson, Christian Nyberg and Maria Kihl
Published in proceedings of the SPIE conference on Performance and Control of the Next Generation Communication Networks, Orlando, Florida, 2003

Paper 4 - Overload Control of a Parlay X Application Server
Jens Andersson, Maria Kihl and Daniel Söbirk
Published in the proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems, San Jose, 2004

The following research papers do not appear in the thesis

Paper 5 - Convergence-Analysis of the Internet and the Telecommunication Architectures
Jens Andersson, Shabnam Aprin and Maria Kihl
Published in proceedings of the 16th Nordic Teletraffic Seminar, Helsinki, 2002

Paper 6 - Priorities and Overload Control in OSA
Jens Andersson, Maria Kihl and Christian Nyberg
Published in proceedings of the IEE Fourth International Conference on 3G, London, UK, 2003

1. Background

Since the penetration of the Internet, the development of new services in the communication area has accelerated. For a long time the only widely spread way of instant real time communication over distances has been telephony, but Internet provides alternative ways of communication. The Internet is still not capable of delivering speech at the same Quality of Service (QoS) as the telephony networks, but it is getting closer. The Internet has grown very fast and turn over enormous amounts of money. The development of the telecommunication networks has been very slow for a long time. The question that the telecommunication operators, among others, have asked recently is how the Internet could reach such penetration. This is a research area itself, and it cannot be answered shortly. A lot of factors have contributed to the success of Internet, see [1].

One of the great opportunities with Internet is how it reaches out to the users with all its content, applications and services. The most common access point to the Internet is the Personal Computers (PCs). Almost all computers are equipped with a web browser, and the Graphical User Interface (GUI) makes the Internet easy to use. Another factor that has had impact on the penetration of the Internet is the many providers of services and applications. Skills in software development is very common nowadays. Many software developers correspond to faster development pace and a wider range of services and applications.

The main service of the telecommunication networks have been to establish a call between two parties and deliver speech between them. The few extra services provided in the fixed telephone network, such as connecting a third part to a call, are not known for their user-friendliness. The mobile operators have been a bit faster to develop new services and applications and above all the Mobile Terminals (MTs) usually provides a better GUI for usage of their applications and services. They have seen the possibility to increase the revenues. An example of a successful service is the Small Message Service (SMS), which have had a great penetration.

It is foreseen that if the 3G telephone networks should become successful, it is necessary that attractive services and applications are delivered. The first release of UMTS resembles the GSM/GPRS system, only differing in the first radio access network stage where Universal

Terrestrial Radio Access Network (UTRAN) was introduced. UTRAN offers improvement of the bandwidth and the number of simultaneously users. Customers are not interested in a new network if they do not see any difference. The provided bandwidth must be utilized by innovative services and applications.

Until recently the development of new applications and services in the telecom networks has only been feasible for people inside the network with deep knowledge of telecommunications. However, new methods and service architectures have been developed to increase the pace of development of new services in telecommunication networks. The trend is to let software developers make use of the network resources via Application Program Interfaces (APIs). Examples of such architectures are OSA [2], Parlay [3] and JAIN [4]. The APIs are provided by a gateway, and several Application Service Providers (ASPs) use the same gateway. Parlay and OSA is more or less the same specification and are often referenced to as Parlay/OSA. Also in this thesis that notation will be used.

To protect the processing nodes in a service architecture, overload control mechanisms must be applied. Research on overload control in earlier service architectures is an old item. Former service architectures are described in Paper 1 and references to research results considering overload control in former service architectures can be found in the Paper 2 and Paper 3. This thesis is focused on overload control in a Parlay/OSA environment.

To summarize, the telephone operators want to increase their revenue. Inspired by the ease to use, the development pace of new services and turnover in the Internet, telecom operators have tried to improve their service architectures. The new proposals need overload control mechanisms to be able to maintain QoS when there are too many applications that want to make use of the same network resource at the same time.

2. Services and Applications

The structure of service architectures has changed over time. Paper 1 included in this thesis gives a good overview of the development of the service architectures. The definition of a service and an application and the difference between them is sometimes diffuse. An application is the Interface to the users. It is the application that facilitates for a user to make use of a network. In this thesis a service means something provided by the network. Example of a telecom service may be to establish a call. An example of an application might be an application initiated call, where two parties can be connected by clicking a link. Of course an application does not have to use any telecom network capability, but in this thesis it is only applications that do which are of interest.

The feasibilities for creation of new applications and services differ dependent of which telecommunication network that is considered. Three main classifications of application and service creation technologies can be identified. First, the Intelligent Network (IN) concept that has been a successful architecture. Second the Parlay concept which is foreseen to introduce a lot of new applications and services. The third technology is the mobile terminal type, which is used in mobile networks. The IN concept provides services from inside a network. Therefore the IN concept provide complete security but is complicated from development aspects. Parlay architectures provide feasibilities for applications to reside outside a telecom network and to make use of network resources like call control or user location. In this way a much easier service creation environment is provided. The third concept is so far only used in the mobile telecom networks, as it requires a mobile terminal. The applications can be created for instant execution on a mobile device. For instance J2ME (Java 2 Micro Edition) [5] can be used for such creation. The mobile terminal type of application will not be involved in further discussions of this thesis, it can just be mentioned that these will co-exist with the services built on IN technology and those built on Parlay technology. Applications connecting the different service technologies are also likely.

However, experience show that also service developers that use Parlay/OSA need knowledge of telecommunications. Thus, to simplify the art of application development even more, the Parlay X Web Services [6] have been introduced.

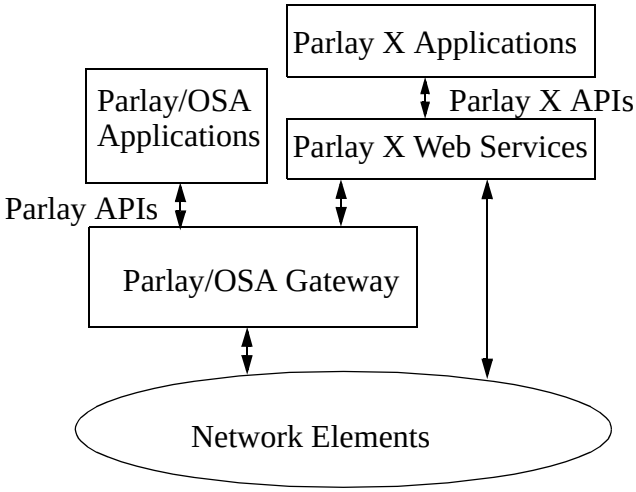


Figure 1 The relation between Parlay X applications and regular Parlay applications

2.1 Web Services

Web services is a concept often used in the Internet, see Menascé et al. [7]. By using high level protocols based on the Extensible Markup Language (XML), advanced service requests can be created. The applications that make use of the Web Services can be written in for example java or Visual Basic or it can be an XML script. With Parlay X Web Services Service Providers (SPs) can provide advanced telecom services to applications by use of a single Simple Object Access Protocol (SOAP) request. Thereby the potential number of application developers are increased.

The Parlay X Web Services specification [8], specifies a set of Web Services. When an application wants to make use of a Parlay X Web Service for example SOAP is used to specify what kind of service that is called. Figure 1 shows an abstracted view of the relation between Parlay X applications and regular Parlay/OSA applications. As seen in the figure it is feasible for the Parlay X services to have instant communication with the Network Elements, but this is rare and this thesis is focused on communication via the Parlay/OSA gateway. So when the SOAP messages are sent to the server hosting the Web Services the server call the Parlay API to complete the service call.

SOAP is a protocol based on XML and often transported by Hyper Text Transfer Protocol (HTTP), see Nielsen [9] et al. SOAP can be

used in combination with other protocols as well, but HTTP is common in the context of Parlay X Web Services.

3. Overload Control

In the context of overload control not only the action to take during overloaded situations should be considered, but also preventive measures. Often overload control is divided into Load Balancing and Admission Control. The Load Balancing algorithms try to share the load in a distributed system such that the different entities are exposed to about the same load. The aim with the admission control is to reject requests for service in the cases when the load cannot be balanced such that an overload situation can be avoided. Load balancing is only interesting when a distributed system should be protected.

To achieve a well performing overload control one important issue is the reliability of the load information. The methods of measuring the load is dependent of the feasibilities in the system. Examples of some common methods to measure current load are

- Queue length measurements; Based on the number of job in the job buffers conclusions of current load and waiting times are made, see Voigt et al. [10].
- Call count control; By counting the number of accepted requests for service during a time interval, the current load is predicted.
- Response time measurements; When the service of a request results in a response, the time can be measured. If the service time gets too long, the system can be assumed to be overloaded. This method is used in for example paper 4 in this thesis.
- CPU load measurements; A common method in the area of web servers is to measure the CPU load when the overload control mechanism is placed on the same entity as the web server. Cherkasova et al. [11] presents an example of this.

Dependent on the design of the system, the different measurement methods have different advantages. Performance of the Load Balancing and Admission Control algorithms are also affected by the design of the system. Examples of circumstances that have impact on the performance of overload control mechanism are:

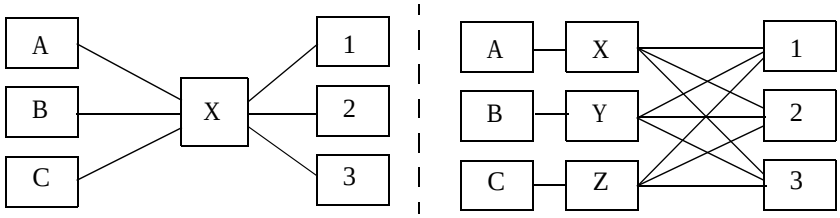


Figure 2 Examples of overload control mechanisms to protect a distributed system. The X, Y and Z boxes correspond to overload control boxes.

- Delay of load information; The information about current load status of the protected system can be delayed. For example there might be an inertia in the system or the load status might be updated once every time unit. The topic of old load information is treated by Dahlin [12] and by Mitzenmacher [13].
- Messages of different priorities; Overload control mechanisms in an environment with different priorities require methods to distinguish between requests to give special treatment. In Berger et al. [14] priorities are taken into account.
- Distributed environments where different nodes have different capabilities; If a system like a Parlay/OSA gateway is distributed, it is likely that some of the Service Capability Servers cannot treat call control messages and another cannot treat charging.

An overload control mechanism may appear in a lot of configurations. Figure 2 shows two possibilities. In the left configuration all sources (*A*, *B* and *C*) are connected to the same overload control mechanism (*X*), which protects the system (*1*, *2* and *3*). In the right configuration all sources (*A*, *B* and *C*) have their own mechanism (*X*, *Y* and *Z*). The right configuration could for example be applied to an environment including contracts where node A, B and C have different constraints of number of accepted service requests. The left configuration could for example be used by a system where node 1, 2 and 3 are extremely sensitive to overload, since the load control mechanism *X* has total control of all incoming requests.

3.1 Load Balancing

There exist several algorithms to deal with load balancing. Which algorithm that is the best is totally dependent of circumstances like those mentioned above. A rule of thumb is to keep it simple. Very often advanced algorithms show slightly better performance to the price of complicated implementation and advanced calculations that must be performed. A short introduction to the most common load balancing algorithms follows below.

Least load

The least load algorithm is simply to always send requests to the least loaded node. This method requires a good method for load measurements. In the configuration shown to the right in Figure 2 node X, Y and Z might act in an oscillating way if the load information is updated in time intervals.

Round Robin (RR)

Round Robin is a classical method where the requests are distributed according to a repetitive pattern. In Figure 2 the pattern would look like 12312312... A variant of RR is Weighted Round Robin (WRR) used where for example the capacity of the nodes might differ. Assume that node *1* has twice as much capacity as the other processing nodes, then the pattern would look like 12131213...

Random selection

For each arriving request the load balancer chooses a node randomly. Also the random selection method can be weighted such that the choice of a certain node is more probable than another.

3.2 Admission Control

The admission control can be dealt with in several ways. Dependent of the environment, different algorithms fit differently well. In the architectures considered in this thesis, contracts about acceptance rate and time constraints are common. Therefore, algorithms that can deal

with guaranteed values are advantageous in these cases. A short introduction to the most common admission control algorithms follow below. Several variants of each algorithm are common, but the basic ideas are presented in this section.

Percent blocking

A ratio of how many of the arriving messages that should be rejected is set. The ratio is somehow connected to the load measurements. The percent blocking algorithm is evaluated in Berger [15].

Call gapping

Time is divided into intervals. During each interval a certain number of requests for service is accepted. If additional arrivals occur they are rejected, see [15].

Token bucket

The token bucket algorithm is developed to reduce burstyness. Tokens arrive at a bucket with a certain rate. A request is accepted if there are tokens in the bucket at arrival. Each accepted request decrease the number of tokens with one.

Leaky bucket

The leaky bucket algorithm totally eliminates the burstyness. Accepted messages are stored in a finite queue. Each time interval the queue sends a message according to FCFS scheduling algorithm. If the queue is full on the arrival of a request, the request is rejected.

4. Summary of Papers

In this section follows short summaries of the papers included in this thesis. The main research question that a paper investigates is highlighted.

Paper I

The main purpose with this paper is to describe the development of the service architectures. A description of Parlay and OSA and the creation of them is given. It is also shown alternative solutions of how IN services can be reached from Internet, but the paper points out the advantages of Parlay/OSA and a standardized solution. Also some performance aspects and potential bottlenecks are highlighted in a Parlay/OSA architecture.

Paper II

This paper investigates different methods to measure the load in a non-distributed OSA gateway. Admission control algorithms and their cooperation with the measurement methods are also investigated. The complete overload control mechanism is proposed to follow the guidelines for overload control in the OSA standard. The proposed overload control mechanism considers messages of one kind and with one time constraint.

Paper III

In this paper priorities and time constraints for different service requests are introduced. An overload control mechanism for Parlay/ OSA in line with the guidelines in the standards is presented. Paper III proposes to use the shortest deadline first (SDF) scheduling algorithm in the gateway to manage the time constraints. When SDF is used an algorithm is proposed which predicts if overload will occur, and acceptance or rejectance of requests from a certain priority is thereby decided. The paper proposes how the priorities can be set to optimize revenue for the operators. In this paper a non-distributed gateway is assumed.

Paper IV

This last paper investigates overload control in a Parlay X web services environment. An overload control mechanism is proposed to deal with messages corresponding to an application providing application initiated calls. Messages of three kinds with different priorities are considered. The mechanism also support guaranteed values of

minimum number of accepted calls per time unit. This paper considers a distributed environment where robust algorithms are proposed for overload control. The admission control algorithm tries to keep short waiting times in the processing entities, but remain high utilization by increasing waiting times to decision if a message should be rejected or not.

5. Further Research

I will continue to investigate load balancing and admission control mechanisms for web services, as investigated in paper 4. Both load balancing and admission control can be optimized favourable to different variables. It should be interesting with a deeper investigation regarding optimal decisions for profit optimization.

Another topic that needs more research, which I also will pay attention to is the contract writing. How should the contracts be written for optimal performance and fairness. In Paper 4 we briefly express how important the choice of time base used in the contracts is.

Reference

[1] Ericsson and Telia, Understanding Tele Communications 2, Studentlitteratur, 1998

[2] http://www.3gpp.org

[3] www.parlay.org

[4] www.java.sun.com/products/jain

[5] http://developers.sun.com/techtopics/mobility/j2me

[6] White paper, “Parlay APIs 4.0; Parlay X Web Service”, http://www.parlay.org, December 2002

[7] D. A. Menascé and V. A. F. Almeida, “Capacity Planning for Web Services”, Prentice Hall, 2002

[8] Parlay X Web Services, specification v1.0, http://www.parlay.org, May 2003

[9] H. Nielsen, P. Leach and S Lawrence, “An HTTP Extension Framework”, IESG Networking Group, RFC 2774 (Experimental), February 2000

[10]T Voigt and P Gunningsberg, “Adaptive Resource-based Web Server admission Control”, ISCC seventh symposium on Computers and Communications, 2002

[11]L Cherkasova and P Phaal, “Session-Based Admission Control a Mechanism for Peak Load Management or Commercial Web Sites”, IEEE Transactions on Computers, vol. 51, 2002

[12]M Dahlin, “Interpreting Stale Load Information”, IEEE transactions on parallel and distributed systems, vol. 11, no 10, 2000

[13]M. Mitzenmacher, “How useful is old information”, IEEE Transactions on Parallel and Distributed Systems, vol. 11, 2000

[14]A. Berger and W Whitt, “Workload bounds in fluid models with priorities”, Performance evaluation 41, 2000

[15]A. Berger, “Comparison of Call Gapping and Percent Blocking for overload control in distributed switching systems in telecommunications networks”, IEEE Transactions on Communications, vol. 39, 1991

PAPER I

Service Architectures for the Next Generation Networks: an Overview and Some Performance Aspects

Maria Kihl and Jens Andersson

I

Abstract

In the next generation networks, 3G-networks and so forth, it is foreseen to be a convergence between the different networks. The telecommunication networks will deploy a new kind of service architecture. The new service architectures shall be developed to enable access of a network's capabilities from other networks. The same applications should be able to be deployed in any telecommunication network. This article contains an overview of the former, current and proposals of future service architectures. Extra attention is paid to describe Parlay and Open Service Access (OSA), two of the most promising service architectures. Further, we discuss some performance problems that may occur when implementing the architectures in a real network.

1. Introduction

Today, both mobile networks and the Internet are widely used and the number of applications is increasing. In many ways, both architectural and the purpose they serve, the networks are converging. As the different networks converge, the networks are expected to deliver

about the same services and applications. This, together with the fact that there is an increasing demand for new services and applications, has lead to an evolution towards so called open service architectures. An open service architecture provide the applications with abstracted network capabilities via standardized APIs. Therefore the pace of development of new applications should increase as a much easier service and application creation environment is provided. Another benefit is that the open service architecture provide an application to be deployed to any network supporting the same service architecture. However, the open service architectures is not the only way to access services from other networks or increase the pace of creation of new services and applications. Alternative proposals exist and will be presented later in this paper.

Several architectures to improve service creation have been proposed by different organisations. TINA was one of the first proposals, with the intention to provide building blocks to create new services. *Parlay*, developed by the Parlay group [1], is an example of a proposal of an open architecture. For example by using the Application Program Interfaces (APIs) defined in Parlay, PSTN-services may be reached and controlled from the Internet. *Open Service Access* (OSA) is an architecture developed by the Third Generation Partnership Project (3GPP) [2]. It is based on Parlay, but is developed especially for 3G-networks. The purpose of OSA is to enable access to network capabilities from the 3G-networks via APIs.

Many papers have described their solutions on how different networks should be converged. Hubaux *et al.* [3] contains an overview of different solutions to connecting Internet with PSTN. Moerdijk and Klostermann [4] describe Parlay and OSA. Moiso and Sommantico [5] discuss the architecture and advantages of OSA. Chapron and Chatras [6] discuss the feasibility of accessing IN-services from a packet-switched network. Licciardi *et al.* [7] contains a general discussion about the advantages with open Application Interfaces (APIs) used in both Parlay and OSA. Wang et al. [8] present a good overview of which protocols that are used when two entities are communicating both within an Intelligent Network and within an IP Network and also between an IN and an IP network.

The purpose of this article is to describe the former, current and proposals of future service architectures and how different networks are proposed to communicate with each other. In particular we

examine the feasibility of accessing services in a 3G-network from Internet.

2. Service Architectures

In this chapter we present the service architectures IN, TINA, Parlay and OSA. One of the advantages of Parlay and OSA is that these will introduce the ability for applications residing outside a network to make use of the network's resources. Some alternative solutions are presented of how IN services can be reached from the Internet, as an example to show that it is feasible without Parlay and OSA.

As OSA is like a slightly modified Parlay architecture we have chosen to only give a thorough description of OSA.

2.1 Intelligent Networks

The objective of the Intelligent Network (IN) is to provide value-added services to the users in a PSTN. The IN is built on the simple idea of separating the service logic from the switching nodes. In this way the operators are offered a much more scalable service platform. Earlier it had been necessary to implement a new application in all switching nodes, but with IN and its centralized architecture the time for deployment could be decreased.

Figure 1 shows an overview of the IN service architecture. The nodes which hosts the applications are called Service Control Points (SCPs) and the switching nodes that deals with the switching and the call control are called Service Switching Points (SSPs). The Local Exchanges (LEs) forwards the "requests" from the users until they reach an SSP. The Service Management System (SMS) contains functions that enable management of the IN system. Another important entity in the IN architecture is the Service Creation Environment (SCE) that contains service development tools which the operator uses for creating new services and applications. If an application needs to work with any kind of data there is a Service Data Point (SDP), which is a database that holds information related to the services.

Typical IN services are local number portability, credit card calling, toll-free numbers and so forth. When any of the supported services are requested, the LE connected to the user recognizes this. The LE

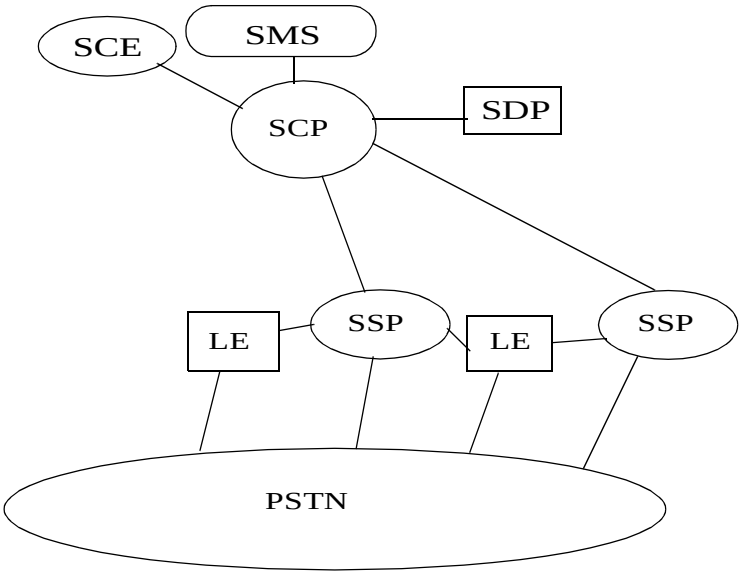


Figure 1. Overview of the Intelligent Network Service Architecture

contacts the nearest SSP, which opens up a dialogue with the SCP that executes service logic corresponding to the requested service. The communication between different nodes such as SCP and SSP is performed via the Common Channel Signalling System No. 7 (SS7) [9] which is a global standard defined by the ITU-T. The SS7 protocol stack contains a couple of different protocols with different advantages. In cases when IN service control is needed, the SSP can trigger the SCP by use of the IN Application Part (INAP) protocol. If an SSP communicates with another SSP they use the ISDN User Part (ISUP).

A mobile version of IN has also been designed for GSM networks. It is called Customized Applications for Mobile network Enhanced Logic (CAMEL). The equivalent to INAP in IN is CAMEL Application Part (CAP) in CAMEL. CAMEL is for example described in Guelen and Hartman [10].

Accessing IN-services from IP-networks

There have been several proposals for how to access IN-services from the Internet. Since long, much work has been performed on how to access IN-services from a VoIP network. That VoIP is becoming more

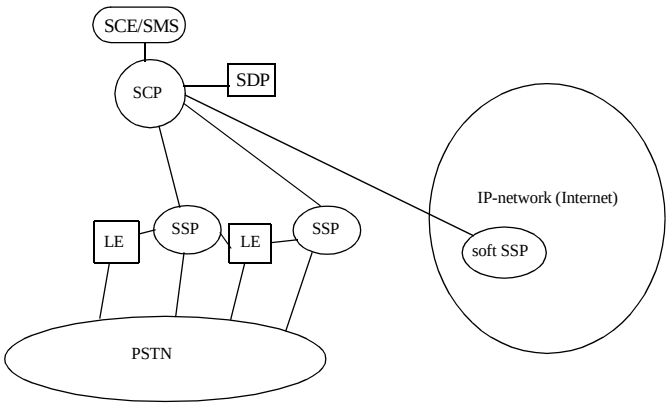


Figure 2. Accessing IN from an IP-network using a soft switching point.

and more common makes it an even more interesting topic. Thus, there are a lot of documentation and many ideas on accessing the IN-services from VoIP protocols SIP and H.323.

One common solution is a so-called soft switch which acts like a gateway, communicating with the SCP in the IN, see Figure 2. This soft switch must appear just like an ordinary SSP for the SCP although it is using IP- based protocols instead of SS7. The service requests arriving at a soft switch may be coded using higher layer IN-protocols, like INAP or TCAP, but one or more of the lower layers will be replaced with TCP/IP (see Ciang *et al.* [11]).

Similar solutions are described by, for example, Wang et al. [12] that have a suggestion of a soft switch that can access IN-services from a SIP network, denoted an Intelligent Services Gateway (ISG). Dagiuklas and Galiotos [13] have a proposal of how to access the IN-services from an H.323 network.

2.2 The Telecommunication Information Networking Architecture (TINA)

The TINA consortium was created in 1993 and was aimed to define a global architecture for telecommunication systems based on advanced software technology. The TINA architecture consists of numerous objects, each a typical part of a service. By using a couple of different objects (building blocks) it is possible to create advanced telecom services. In this way many more potential service developers could be

reached as the complexity of creating new services should decrease. In the IN, expertise knowledge is required to create new services, as a consequence of the advanced signals and protocols.

The TINA consortium was a very large consortium with many companies involved, but unfortunately the project did not succeed in the way it was planned. One of the reasons might have been that the building blocks are too many and that it is still not possible for any programmer to get the whole picture of which components that have to be used to create a new service or in which order the objects should be called. Another possible reason could be the aspects of real time performance. The realization of a simple service as establishing a call is very complicated and has a critical connection time, see Kihl [14]. However the TINA consortium was a good initiative which was a first step toward the open service architectures.

2.3 Parlay

The Parlay group [1] is a consortium that was founded by British Telecom, Microsoft, Nortel Networks, Siemens and DGM&S Telecom in 1998. This group has focused on the creation of the Parlay specification, which provide easier service and application development than the IN. The specification specifies a set of standard Application Program Interfaces (APIs) that will enable applications residing outside the network to access and control resources inside the network.

The major advantage and one of the reasons why the Parlay project once was created is the ease of creating new applications. The Parlay API specifications are open and technology-independent, so that a wider range of Independent Software Vendors (ISVs) may develop and offer advanced telecommunication services. An API is intended to be simple in order to be applied in all different kinds of networks and can be used from 3rd party application developers. In IN the operator is responsible for the creation and operation of all the applications. Further, Parlay offers better opportunities to test the software before it get deployed.

The Parlay APIs consist of two categories of interfaces, *Service* and *Framework*. The Service interfaces are the interfaces that offer applications access to network capabilities such as call control and

messaging. The Framework interfaces are the interfaces that take care of the security and manageability.

The design and implementation of the API and applications using this API supports a wide range of existing distribution middleware as CORBA, DCOM and JAVA/RMI.

2.4 Open Service Access (OSA)

At first OSA was an acronym for Open Service Architecture, but it has been re-termed to Open Service Access. OSA is based on the concept of Parlay and is developed by the 3GPP. OSA differs from Parlay in a way to better satisfy the needs for a 3G network. The concept is still the same but APIs of no use are removed and new APIs are introduced.

Advantages

One of the reasons why OSA has been evolved is the opportunities seen in the area of applications and wireless Personal Digital Assistants (PDAs). It is foreseen that there will be a great demand for applications and in order to respond to this demand the pace of the service and application development has to speed up. Therefore the OSA has been proposed, to make it easier to develop and test new services outside the telecom domain. The number of feasible service providers has increased because of the fact that OSA APIs offers the security and integrity that the operators need to open up their networks to independent software developers and service providers (see Lundqvist [15]).

Architecture

OSA consists of three parts, namely the *Application Servers* (ASs), the *Framework* and the *Service Capability Servers* (SCSs), see figure 3. The ASs are the entities hosting the applications, the Framework manage the security aspects and the SCSs provide the ASs with Service Capability Features (SCFs). The SCFs can be seen as abstracted network functionality.

Applications are, for example, application initiated call, conferencing and location based applications. The ASs can both reside inside and outside the network where the capabilities reside. An

application can be triggered in different ways. Examples of two common ways are that either an application is triggered by the end-user

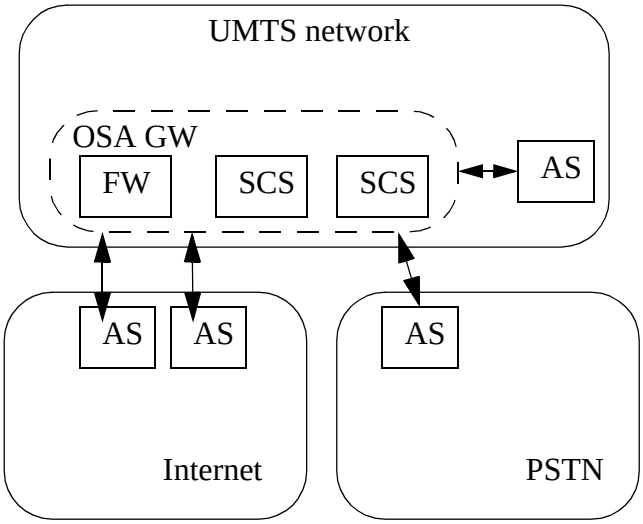


Figure 3. Network architecture for OSA.

dialling a special number just like in an IN, or that the application is initiated by some kind of HTTP request.

The Framework provides the applications with basic mechanisms, like authentication before accessing the network functionalities or discovery of the capabilities needed by an application. The Framework SCFs include Trust & Security Manager, Discovery and Integrity Management SCFs. It is due to these SCFs that OSA can offer the security that is required of the operators. There is always exactly one Framework in an OSA service architecture. The Framework together with the SCSs constitute a so called OSA gateway. The gateway acts as a mediation device offering a uniform and technology independent interface.

The SCSs provide the applications with all the network functionality required to construct services via SCFs. There may be one or more SCSs in an OSA gateway. The SCSs communicate with the network entities (HLRs, MSCs, SCPs etc.) needed to perform a specific SCF. The Network SCFs includes the Call Control, User Interaction, Mobility, Terminal Capabilities, Data Session Control, Generic Messaging, Connectivity Manager, Account Management and Charging SCFs. These different SCFs are the network capabilities that

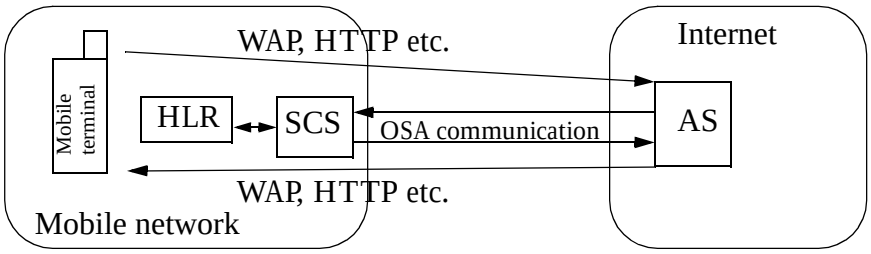


Figure 4. Mobile user accessing a location based application residing in Internet.

the service providers can use as building blocks to compose and create new applications. An SCF can be compared to an object class with some functions. For example, Call Control can be seen as a class consisting of functions like “create call leg”, “delete call leg”, “connect call legs” etc.

The applications developed can be executed on several networks and terminals thanks to OSA that have standardized APIs towards the ASs. A typical scenario that is foreseen is when a mobile user executes an application located in the Internet. For example if an application residing in the Internet makes use of the location of the mobile user, the communication can look like in figure 4. First the user use GPRS to reach the application. The application is then triggered by HTTP or similar protocol. The application use an appropriate SCF to reach the location of the mobile user and can return the appropriate information via gprs.

Just like in Parlay, OSA uses an object-oriented technique, like CORBA, which makes it possible to use different operating systems and programming languages in ASs and SCSs.

3GPP has also introduced an interesting concept called Virtual Home Environment (VHE) [16]. The idea of VHE is to consistently present the same personalised features, User Interface customizing and services to the user, independent of which terminal or network that is present at the moment. More details about VHE and OSA can be found in Moretti and Depaoli [17].

3. Performance Issues

One important issue when developing new service architectures is how to ensure a good QoS for the users. When a user wants to run an application, it is necessary that the delays are short. Therefore, it is important that the potential performance bottlenecks are identified before the architecture is deployed.

When studying Parlay and OSA, one potential bottleneck is easily identified, namely the Gateway or actually the SCSs. Since the Gateway is a centralized service control point it is sensitive to overload. Therefore, overload control mechanisms should be developed in order to maintain a good performance during all traffic situations.

Further, the gateway will provide many types of services. These services may have different priorities that must be taken into account. The services may have so-called QoS contracts that must be kept. High priority services, for example those that generate profits for the service provider, should not be blocked in cases of overload.

Other potential bottlenecks are the ASs in Parlay/OSA. As it is here the execution of the application will take place it must be controlled how many applications that are executed at the same time. Both the ASs and the SCSs may have distributed structure. Distributed systems need load balancing to prevent overload from occurring with current capacity.

So far, there has only been one published paper about performance analysis of the service architectures described above. Melen *et al*. [18] use a simple queuing network model to investigate a Parlay gateway supporting multiple services. They develop a scheduling mechanism for the gateway that ensures that each service obtains processing capacity that is proportional to the offered load for that service.

4. Conclusions

The telephony networks and the Internet will become converged. Customers want to reach the same applications independent of in what network they are positioned or what terminal they use. Also, the demand for new applications and advanced services is increasing as these generate great revenues for the telecom network operators.

Therefore, new service architectures must be developed. Several architectures have been proposed by different organisations to increase the pace of service and application creation. In this article we have described Parlay and OSA, two of the most promising service architectures.

Since both Parlay and OSA are based on distributed processing and open APIs, several performance problems may be identified. Therefore, it will be necessary to develop performance models for these architectures and then investigate how they behave during various traffic scenarios.

References

[1] The PARLAY group home page, www.parlay.org

[2] The 3GPP home page, “www.3gpp.org”.

[3] J-P Hubaux, C Gbaguidi, S Koppenhoefer, J-Y Le Boudec, “The impact of the Internet on telecommunication architectures”, Computer Networks 31, 1999

[4] A Moerdijk, L Klostermann, “Opening the networks with PARLAY/OSA APIs: standards and aspects behind the APIs”, http://www.3gpp.org/.

[5] C Moiso, M D Sommantico, “Identifying strategies for migrating Intelligence to 3G Networks to deliver next generation value-added services”, http://exp.telecomitalialab.com/mobile_art04_p01.htm, 2001.

[6] J-E Chapron, B Chatras, “An analysis of the IN call model suitability in the context of VoIP”, Computer Networks 35, 2001

[7] C-A Licciardi, G Canal, A Andretto, P Lago, “An architecture for IN-internet hybrid services”, Computer Networks 35, 2001

[8] W. Wang, Y. Hao, C. Sun, M. Lu and S. Cheng, “ISA: A Stand Alone Services Agent Supporting IN/IP Integration”, IEEE Intelligent Network Workshop, 2001.

[9] “SS7 Tutorial”, *www.pt.com*, 2000-2001.

[10] E Guelen, J Hartmann, “Open service provisioning in GSM -What do we gain with CAMEL, http://www.jens-hartmann.de/papers/epmcc97.pdf, 1997.

[11] T-C Chiang, J Douglas, V Gurbani and W Montegomery, “IN Services for Converged (Internet) Telephony”, June 2000

[12]W Wang, S Cheng, G Bochman, “Accessing Traditional Intelligent Services From SIP Network”, Info-tech and Info-net, 2001
[13]T Dagiuklas, P Galiotos, “Architecture of an enhanced H.323 VoIP Gateway”, Communications, 2002
[14]M Kihl, Overload control strategies for distributed communication networks, Phd thesis, Lund University, Department of Communication Systems
[15]A. Lundqvist, www.incomit.com, May 2001.
[16]3GPP Technical Specification 23.127 v. 3.0.0 “Virtual Home Environment/ Open Service Architecture”, http://www.3gpp.org/ftp/specs/march_00/23_series/23127_300.zip, march 2000
[17]L Moretti, R Depaoli, “OSA Enabled Global Application Roaming”, Proc. of IEEE Intelligent Network Workshop, 2001.
[18]R. Melen, C. Moiso and S. Tognon, “Performance Evaluation of a Parlay Gateway”, Presented at the International Conference on Intelligence in Next Generation Networks, 2001.

PAPER II

Performance Analysis and Modelling of an OSA Gateway

II

Jens Andersson, Christian Nyberg and Maria Kihl

Abstract

It is foreseen that you in the future should be able to use the same applications independent of where you are positioned or which terminal that is used. The open service architectures provide these opportunities. Open Service Access (OSA) is an example of such an architecture and it is part of the specification delivered by 3GPP. This paper explains the OSA architecture and presents a model of an OSA gateway. Further, it discusses and proposes some feasible overload control mechanisms for the gateway. The behaviour of the mechanisms is investigated through simulation.

1. Introduction

During the last years there has been a change in service architectures towards so called open service architectures. One of the first open service architectures that was successfully developed is PARLAY specified by the Parlay group. In Parlay a set of standard application interfaces (APIs) is defined. These will enable applications residing outside the network to access and control network resources.

Open Service Access (OSA) is the service architecture that is proposed for the 3G networks. OSA is based on the concept of PARLAY, and is

developed by the 3GPP [1]. It is foreseen that there will be a great demand of services and in order to respond to this demand the pace of the development has to speed up.

One common problem for all service architectures is what actions to take if the control nodes become overloaded. Overloaded nodes leads to long waiting times for service. If the waiting times get too long, customers will abandon the request for service and perhaps make a retry. These abandoned requests consume valuable processing time. In the worst case, an overloaded node will only be processing abandoned requests for service. Thereby the need of an overload control mechanism is obvious.

Overload control has been around for some decades. In Wildling [2] the protection of telephone exchanges is discussed. One paper on overload control in IN is Kihl [3].

Very few papers have been published on load issues for open service architectures. However, the performance of a Parlay gateway is analysed in Melen [4].

In this paper, we investigate overload control mechanisms for the OSA service architecture. We propose a queuing model for the most critical nodes in the architecture and investigate different ways of measuring load and rejecting customers.

The paper is organized as follows: in section 2 a description of OSA is given. In 3 the simulation model is presented. The proposals for overload control mechanisms can be found i section 4 followed by the results and discussion in section 5. Finally we draw some conclusions in section 6.

2. Open Service Access (OSA)

From the beginning OSA was an acronym for Open Service Architecture, but it has been re-termed to Open Service Access. OSA offers an increased security and integrity enabling the operators to open up their networks to independent software developers and service providers. Thereby the number of feasible service providers has increased.

2.1 Architecture

OSA consists of three parts, the Application Servers (AS:s), the Service Capability Servers (SCS:s), and the Framework. Figure 1 shows one possible configuration of an OSA architecture. The part referred to as the

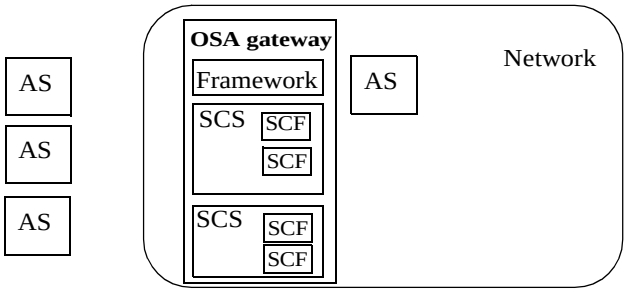


Figure 1. The architectural picture of an OSA architecture

OSA gateway can be built on several physical entities. In Figure 1 the Framework and both the SCS:s constitute the OSA gateway.

The AS:s host the applications. An application is usually triggered by the dialling of a special number or by some kind of HTTP request. The AS:s can be physically positioned inside or outside the network they are communicating with. An example of a typical OSA application in a 3G network is an “application initiated call” proposed in [5]. The sequence diagram of this service is shown in Figure 2.

In an OSA architecture there can be one or several SCS:s, see Stretch [6]. The SCS provides network functionality to the applications via one or several SCF:s. An SCF consists of several narrow functions, which together makes it possible to utilize the network capability. Examples of SCF:s are Call Control, Mobility and Charging SCF. For example the Call Control SCF provides functionality to establish different kinds of calls to a mobile user.

The Framework can be seen as a separate SCS providing the applications with basic mechanisms, like authentication before accessing the network functionalities or the possibility to find out which SCF:s that are provided by the SCS:s. It is important to notice that there is always exactly one framework in an OSA gateway.

2.2 Overload Control in OSA

In an OSA architecture the AS:s and the SCS:s are especially sensitive to overload. It is possible for both the AS:s and for the SCS:s to have overload control.

The overload related functionality is managed by the Framework as described in the specifications [7]. Information about the load condition

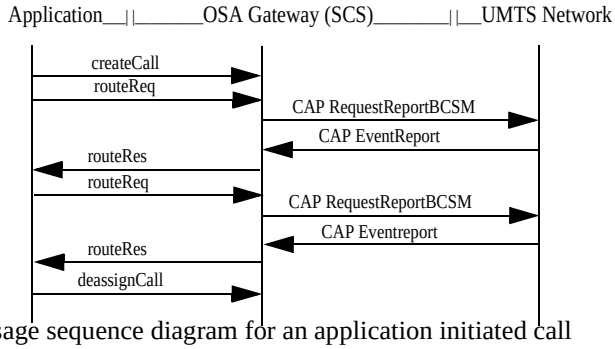


Figure 2. Message sequence diagram for an application initiated call

in the SCS:s and the AS:s can be exchanged between the AS:s and the Framework. This gives the opportunity to control the load either from the AS or from the Framework.

There are three load levels, 0, 1 and 2 corresponding to normal load, overload and severe overload respectively. Nothing is said about how the load levels should be set or what actions they should cause. The actions should be defined in the load management policy, which is created via contract writing.

3. Simulation Model

We have developed a model consisting of one AS and a gateway containing one SCS and a Framework, Figure 3. The gateway can be modelled as only the SCS, as framework requests are assumed to be rare. In the AS the application described in Figure 2 is implemented. Of course in a real system there will be many applications with different behaviour. However, one is enough to create an overload situation and to evaluate the behaviour of an OSA gateway. The arrivals of the application calls are modelled as a Poisson process with the rate λ calls each second. The SCS is modelled as a one server queue with capacity of serving 100 application calls per second. The capacity of the AS is dimensioned so the overload will appear in the SCS and thereby the AS can be seen as a delay.

In Figure 2 it is shown that each service has to execute in the network twice. The first time is modelled as a delay of 10 ms and the second is modelled as an exponentially distributed delay with mean 2 s. The other service times are set as follows: If a message in the SCS results in a new message the execution time is 2 ms, else 1 ms. Each delay in the AS in Figure 3 is 1 ms.

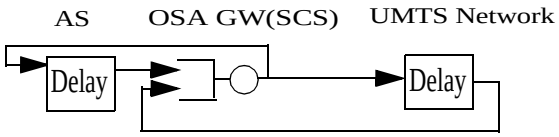


Figure 3. The simulation model

4. Overload Control Mechanisms

An overload control mechanism should measure the current load and reject new calls if necessary. In our model the rejection mechanism is positioned in front of the gateway.

When the gateway is overloaded the waiting times get too long. In [8] a maximal delay of 100 ms is proposed and that value will be used here. If a completed application call has had a mean delay in the OSA gateway longer than 100 ms, it is said to be an expired call.

The main objectives for the overload control mechanism in this paper are to maximize the throughput and minimize the number of expired calls. To do this the number of calls in the gateway should fluctuate as little as possible so that the server is kept busy as much of the time as possible at the same time as the queue length should be kept short. The current load is expressed by one of the three load levels according to the specification, see section 2.2.

4.1 Measurement Methods

Two ways of measuring the load, A and B, are proposed below. In both cases, the measured load at an arrival is compared to a threshold value corresponding to the current load level. If the measured load is above the threshold value at five consecutive arrivals or departures the load level is increased (if smaller than 2). If it is below the threshold at five consecutive arrivals or departures it is decreased (if it is larger than 0).

Method A measures the total number of application calls in the SCS, network and AS. Method B measures the number of calls in just the SCS. Calls in the network and AS will sooner or later come back to the SCS and demand processing. Method A takes this into account, B does not.

To estimate the threshold values when A is used we set T_{tot}=E(total time in system for an application call), T_{scs}=E(total time in the SCS) and

x_{scs}=E(service time in the SCS for each application call in ms). If $\hat{A}$ is the threshold it must satisfy

$$\hat{A} \cdot \frac{T_{scs}}{T_{tot}} = \frac{100}{x_{scs}} \Rightarrow \hat{A} = \frac{100 \cdot T_{tot}}{x_{scs} \cdot T_{scs}} \quad \textbf{(EQ 1)}$$

to satisfy the requirement so that calls not are expired.

When method B is used the threshold $\hat{B}$ can be calculated as

$$\hat{B} = \frac{100}{x_{scs}} \quad \textbf{(EQ 2)}$$

4.2 Rejecting Methods

This paper proposes two methods for rejecting calls, the static method and the dynamic method. Both methods use Percent blocking [9] where R_f=P(a call is rejected).

The static method works like this: when load level is 0, R_f is 0, when load level is 1, R_f is set to 0.5 and when load level is 2, R_f is set to 1.

The dynamic method tries to stabilize the measured load just below the threshold. When load level 1 is reached R_f is increased by 0,1. If load level 1 remains after X seconds, R_f is increased one more time etc. If load level 2 is reached, R_f is increased by 0.4 in the same way. If the load level is 0, R_f is decreased by 0,1 every X:th second. Of course R_f must always be in the interval [0, 1].

In our simulations X is 25*E(total execution time in the SCS for one application call).

Table 1. Threshold parameters used in the simulations

Measurement method	Static method load level 1	Static method load level 2	Dynamic method load level 1	Dynamic method load level 2
Method A	190	210	190	200
Method B	40	45	30	35

5. Results and Discussion

The rejecting and measurement methods will be compared in this section. The comparisons will be done with both constant and varying average

arrival rates, λ. The threshold values are chosen such that the fraction of expired services never exceeds 0.5%. The used threshold values are shown in Table 1.

5.1 Comparisons of Measurement Methods

In the steady state case when we let λ keep the same value during a long interval method B gives a better throughput. We also conclude that method A is more sensitive to changes of the threshold values. If the thresholds are lowered to decrease the rate of expired calls there is a sharp decrease of the throughput when method A is used.

However, steady state arrival rates are not very realistic. It seems more probable that the value of λ is rather bursty and shifts at random times. Figure 4 shows the simulation results when λ is randomly varying between the discrete values 0, 50, 100 and 150 calls per second and the times between changes of λ are exponentially distributed with mean 2.0 seconds. The upper plot shows how λ is varying over 200 seconds with the mean 81.6 calls per second. In the plots it can be discerned that method B is better than method A, because of the smaller variations in the number of application calls in the SCS. The mean throughput values corresponding to each of the graphs starting from the upper are 60.3, 57.6, 63.5 and 60.4 respectively.

5.2 Comparisons of Rejecting Methods

In the steady state case the throughputs are about the same irrespective of which rejecting method that is used. However, when λ shifts it can be discerned that the static method has a better behaviour from transients point of view in Figure 4 When there is a change from $\lambda = 0$ to $\lambda = 150$ it can be discerned how the dynamic method has a slow reaction.

This means that the static method fulfils the requirements just as well as the dynamic method and it seems to have a better behaviour concerning transients.

6. Conclusions

In the OSA architecture it has not been defined how to measure the load or how to react on an overload situation. In this paper it is shown that the

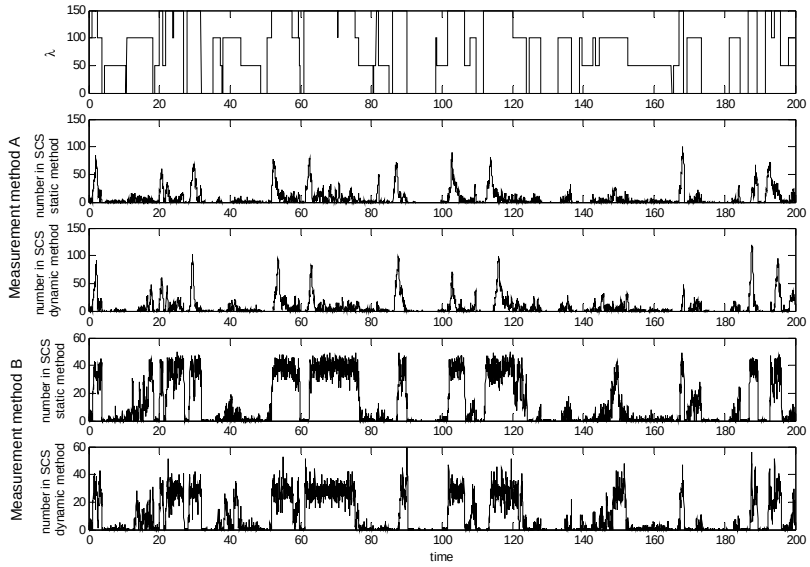


Figure 4. The number of services in the SCS is plotted as a function of time when λ is varying as the top plot. In each plot different measurement method or rejecting method is used

throughput is larger when the number of calls in the SCS is used as a measure of the load than when the total number of active calls are used as a measure. We have also proposed two rejection methods of which the static method seems to have the best behaviour.

References

[1] The 3GPP home page, "www.3gpp.org"

[2] Wildling, K., T. Karlstedt, “Call Handling and Control of Processor Load in SPC-systems”, ITC 9, Torremolinos 1979

[3] Kihl, M, Nyberg, C, "Investigation of overload control algorithms for SCPs in the intelligent network", Communications IEE Proceedings, 1997

[4] Melen, R., Moiso, C., Tognon, S., “Performance evaluation of a Parlay gateway”, "http://exp.telecomitalialab.com/pdf/06-MOISO4.pdf", 2001

[5] ETSI standard 201 915-4 v1.3.1 “OSA API; Part 4: Call Control SCF”, july 2002

[6] Stretch, R.M., “The OSA API and other related issues”, B T Technol J. Vol. 19 No 1, Jan 2001

[7] ETSI standard 201 915-3 v1.3.1 “OSA API; Part 3: Framework”, july 2002

[8] Eurescom Technical Information, “Non-functional aspects and requirements related to Parlay/OSA products”, june 2002

[9] Berger, A.: "Comparsion of Call gapping and Percent blocking for overload control in distributed switching systems and telecommunications networks", IEEE Transactions on Communications, vol. 39, 1991

PAPER III

Performance Analysis and Overload Control of an Open Service Access (OSA) Architecture

Jens K. Andersson, Christian Nyberg and Maria Kihl

III

Abstract

The trend of the service architectures developed in telecommunications today is that they should be open in the sense that they can communicate over the borders of different networks. Instead of each network having their own service architecture with their own applications, all networks should be able to use the same applications. 3GPP, the organization developing specifications for the 3G networks has specified the standard Open Service Access (OSA), as a part of the 3G specification. OSA offers different Application Program Interfaces that enable an application that resides outside a network to use the resources of the network. This paper analyses the performance of an OSA gateway. It is examined how the overload control can be dealt with in a way to best satisfy the operators and the 3'rd parties. There are some guiding principles in the specifications, but a lot of decisions have to be made by the implementors of application servers and OSA gateways. Proposals of different requirements for an OSA architecture exist such as, minimum amount of accepted calls per second and time constraint for the maximal total delay for an application. Maximal and fair throughput have to be prioritized from the 3'rd parties view, but profit is the main interest from the operators point of view. Therefore this paper examines a priority based

proposal of an overload control mechanism taking these aspects and requirements into account.

Keywords: Open Service Access, overload control, quality of service, priorities

1. Introduction

During the last years there has been a change in the evolution of service architectures. Until recently, each network has had its own service architecture and only the operator has been able to create and introduce new services. Today a couple of consortia are developing specifications for service architectures which allow interactions between different networks. Thus, an application in one network can use resources from other networks. Service creation will also be much easier in the new architectures. Earlier only experts could create and deploy new services, as thorough knowledge of telecom protocols was required to be able to create an application inside the network.

Open Service Access (OSA) is the service architecture that is proposed for the 3G networks. OSA is developed by the 3GPP [1]. With OSA it becomes easier to develop and test new services outside the telecom domain. Since OSA offers an increased security and integrity, it is possible for the operators to open up their networks to independent software developers and service providers, see Rajagopulan [2].

One common problem for all service architectures is what actions to take if the control nodes become overloaded. To over provision the nodes so that they can cope with all load peaks is too expensive. If a node is overloaded, long queues of jobs will be formed which leads to long waiting times for service. If the waiting times get too long, customers will abandon the request for service and perhaps make a retry. But the abandoned requests also consume valuable processing time. In the worst case, an overloaded node will only work with processing abandoned requests for service.

This problem may be solved by introducing overload control mechanisms in the network. The main idea is to, during overload situations, reject some requests as early as possible, so that the accepted requests receive a good service. To be able to do overload control one needs a way of detecting when a node is overloaded and also a mechanism

that rejects requests when there is overload. There must also be a way of determining how the load measure should be used to calculate the parameters of the rejection mechanism.

Overload control of communication systems has been a research topic for some decades in telephone networks. An early paper is Forys [3] in which the protection of control processors in telephone exchanges is discussed. Some papers on overload control in IN are Pham et al [4] and Kihl et al [5] in which overload control algorithms are suggested and investigated. The general performance of a Parlay gateway, which is almost the same as an OSA gateway was analysed in Melen et al [6]. Overload control mechanisms for an OSA architecture are proposed and investigated in Andersson et al [7].

In the context of overload control, most papers present methods on how to reject new calls in such way that the callers are treated equally. This is, of course, the fairest case from the users' point of view, but the operator's main interest is probably revenue. Therefore, we believe that an overload mechanism based on priorities should be interesting for the service providers. The priority of an application should correspond to the amount of revenue the application generates for the operator. Thus, other variables have to be included in order to maintain the user-perceived Quality of Service of the applications.

In this paper, we give a thorough description of OSA in section 2. We propose a queuing model of an OSA architecture in section 3. In section 4 priorities are discussed. A priority based overload control mechanism is proposed in section 5. In section 6 the simulation parameters are given followed by results and discussions in section 7. Finally we draw some conclusions in section 8.

2. Open Service Access (OSA)

OSA is a collection of open network Application Program Interfaces (APIs) that enable third party vendors to develop and deploy, with the minimum effort, applications that access the full functionality of the underlying network while still preserving its integrity. By abstracting service creation from telecommunication specific details, the development process of new applications is shortened and the creation pace of new applications can be increased.

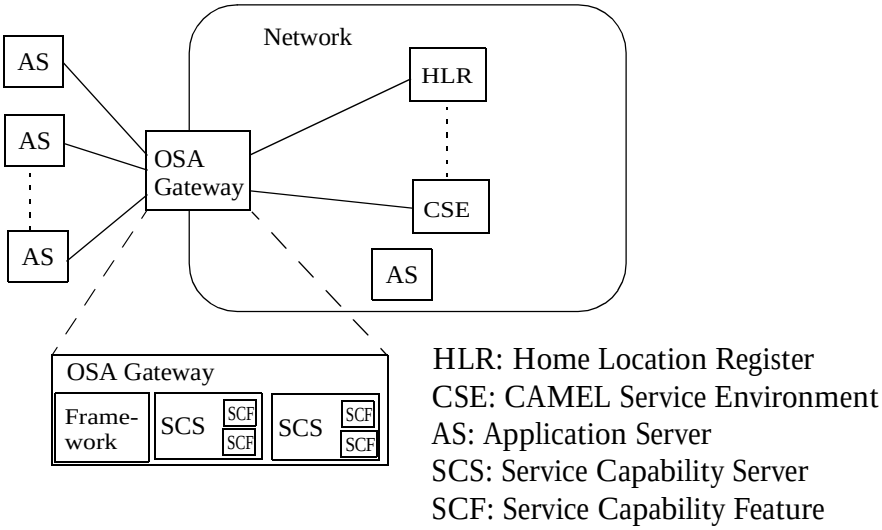


Figure 1. An example of an OSA architecture with a detailed view of the OSA Gateway

An OSA architecture consists of three main parts, the Application Servers (AS:s), the Service Capability Servers (SCS:s), and the Framework. Figure 1 shows one of the possible configurations of an OSA architecture. Each SCS hosts one or several Service Capability Features (SCF:s), which are abstractions of the underlying network functionality. The part referred to as the OSA gateway can be built on one or several physical entities. In Figure 1 the Framework and both the SCS:s constitute the OSA gateway.

The Application Servers (AS:s) host the applications. Each AS can host one or several different applications and provide them with the ability to communicate with the Network via the OSA Gateway. The AS:s can be physically positioned inside or outside the network they are communicating with. An AS positioned outside the network of an operator is typically connected to the Internet. Usually an AS is triggered by the dialling of a special number or by some kind of HTTP request. Examples of applications are conferencing, location based applications and so forth.

In an OSA architecture there can be one or several Service Capability Servers (SCS:s). More about the implementing of SCS:s can be found in Stretch [8]. The SCS provides the applications with network resources via one or several SCF:s. An SCF consists of several narrow functions, which together makes it possible to utilize the network capability. One example

is the Call Control SCF, which provides functionality to connect and establish different kind of calls to a mobile user. Another example is the Charging SCF, which provides functionality to charge the user for a service.

The Framework is the most important part in the OSA architecture. This part takes responsibility for all security aspects of OSA. For example it provides the applications with functionality like authentication before accessing the network functionality or discovery to find out which SCFs that are provided by the SCSs. All the security and integrity functionalities necessary to open up a network are provided by the Framework. It is important to notice that there is always exactly one Framework in an OSA architecture.

2.1 An Example of a Service in OSA

In the specification [9], a couple of OSA applications are proposed. One example is an “application initiated call”. In this application for example a customer accesses a Web page and selects a name on the page of a person or organisation to talk to. The sequence diagram of this application is shown in Figure 2. An application setup consists of a number of OSA messages. First the application sends a createCall message to the OSA Gateway to create objects for further communication. In the Gateway the application call is translated into suitable protocols for communication with for example an UMTS network. When the A party has answered, the application is notified and then the call is routed to the B party.

2.2 Overload Control in OSA

In an OSA architecture there are especially two parts sensitive to overload, the AS:s and the SCS:s. The most critical SCF seen from the aspect of overload is the Call Control SCF which connects and initiates calls. The overload related functionality is managed by the Framework. In the specification [10], there is a description of the functionality that is prepared. Information about the load condition in the SCS:s and the AS:s can be exchanged which gives the opportunity to control the load either from the application side or from the Gateway.

The load condition is described by three levels. Load level 0 corresponds to normal load, load level 1 corresponds to overload and load

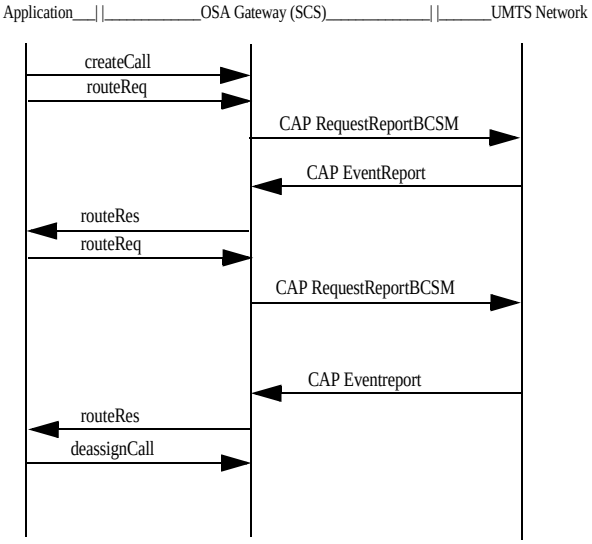


Figure 2. Message sequence diagram for an application initiated call.

level 2 corresponds to severe overload. Nothing is said about how the load levels should be set or what actions they should cause, but corresponding threshold values to load level 1 and 2 can be set. Different SCS:s can have different threshold values for the load levels. The action an overload situation should cause on a specific application is identified in the load management policy, which is created via contract writing (see below).

It is possible for the Framework to subscribe on load information both from an AS and an SCS. The subscription can be either load information sent to the Framework at discrete times or load information sent on a load level change.

2.3 Contract Writing

When a new application is introduced a contract is written with the OSA gateway through the Framework. The contract contains rules and restrictions that should be followed. Proposals of what a contract should include can be read for example in Rajagopulan [2]. A typical contract might include minimum throughput for an application and maximal delay of an application call. Another variable that might be agreed on is the charging criteria. The contracts should not only consist of constraints for the applications according to the gateway. Also the constraints that the application has to fulfil is agreed.

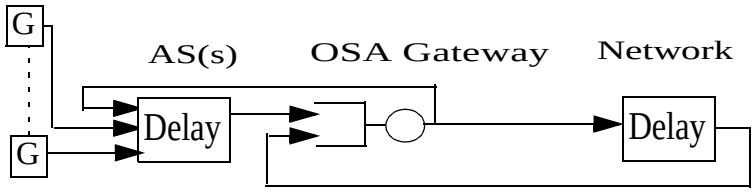


Figure 3. An OSA Model.

3. Model

We have modelled an OSA gateway built on a single physical entity in a multi application environment. Our model consists of *R* applications, a Gateway and a network, see Fig. 3. We do not have to specify how many SCF:s there are in the gateway as they are positioned on the same physical entity anyway.

Each G-box in Figure 3 corresponds to a generator of new application calls to a specific application. We will assume that calls are generated according to a Poisson process or according to an MMPP, but other processes could of course be used in the simulation model. Each application is assumed to be positioned at a non-overloaded AS which is modelled as a delay with deterministic values, depending of which message that should be executed.

An application, a_l, has a guaranteed rate of d_l calls per second, and a total execution time in the Gateway of $x_{tot}(l)$ seconds. Of course the system must be stable when all applications face their guaranteed rate, which implies that

$$\sum_{l=1}^{R} d_l \cdot x_{tot}(l) < 1 \qquad \textbf{(EQ 1)}$$

Each application belongs to a priority, p_j ,$(j = 1 \ldots M)$ and has a time constraint, T_k ,$(k = 1 \ldots N)$. Each priority corresponds to a guaranteed rate of application calls per second. A time constraint T_k corresponds to the maximum delay a message should experience each time it passes the Gateway. In the example in Figure 2 the application call has to pass the gateway five times and if the application call, when it is completed, has had a total residence time in the gateway larger than $5 \cdot T_k$, the call is said

to be expired. The time constraints are set such that $T_1 < T_2 < .... < T_N$. The set of applications with time constraint k is denoted $A(T_k)$. The total guaranteed rate of applications with time constraint k, is denoted λ_k. λ_k is given by

$$\lambda_k = \sum_{l \in A(T_k)} d_l$$

The time constraints and priorities can be set independently of each other for each application.

The gateway is modelled as a single server queue, in which one message at a time is served. This means that all messages are stored in the same queue when waiting for execution. The execution times in the gateway are set to deterministic values, depending of which message that should be executed. We denote with $N(t)$ the number of messages in the gateway with a remaining time less than t before their deadline expires. Another variable that will be used in this article is x_{GW}, which is an estimation of the execution time of a random message in the gateway. We will use different values for this variable, see Section 6

The network is assumed to be non-overloaded and is modelled as a deterministic delay with stochastic elements from, for example, the phone pick up time in Figure 2.

4. Priorities

The priorities should be set in such way that the utility is largest for the priority 1 applications and decreasing by increasing priority levels, as we want to maximize the utility. In this section we propose a utility function that can be used for the settings of the priorities.

4.1 The Utility Function

First of all utility has to be defined. The utility should somehow correspond to the use of spending processor time on an application. We believe that revenue is the main interest for the operator or owner of the OSA gateway, which means that revenue should have influence on the utility. However, the revenue of an application cannot directly be used as a measure of utility. Assume that a very resource consuming application exists. Even if this application generates a good revenue it might happen

that another application exists, which generates less revenue but has a much shorter total execution time in the Gateway. Therefore, the total revenue for the latter application during a minute may be higher than the former. So we have to weight the revenue against the total execution time in the gateway. The utility function of a_l is now described by the following expression

$$U(l) = \frac{r(l)}{x_{tot}(l)} \quad \textbf{(EQ 2)}$$

where *r(l)* is the revenue. *r(l)* is the income the operator receives each time an application call from a_l is served minus the cost of maintaining the network functionality that the application use. In Eurescom Technical Information [11] the players in a Parlay/OSA business environment are discussed. The operator is only one of many players, so the income to the operator per application call is not equivalent to the cost of an application call for the customer as the receipts should be distributed among several players. This means that the utility an application corresponds to is a measure of how much revenue the operator will have through executing applications of the same kind for one second.

Equation 2 gives a good estimation of the utility seen from the aspect of revenue in the short run. But in the long run it should be an advantage for the operator to have an extra goodwill parameter, G, that could be set individually and be added to Equation 2. For example a completely new application representing a new kind of service might lead to larger revenue in the future if it is experienced well and then the development pace of similar applications might increase. So if this goodwill parameter is based on qualified market analysis this would probably increase the revenue in the long run.

The priority of an application depends on the utility function. A discrete number of priorities will be used, and thereby each priority level will correspond to a utility interval.

5. Overload Control Mechanisms

We have developed an overload control mechanism for the gateway, see Figure 4. It consists of a controller, a gate and a selector. The *controller* makes appropriate measurements on the gateway. Also, it analyses the

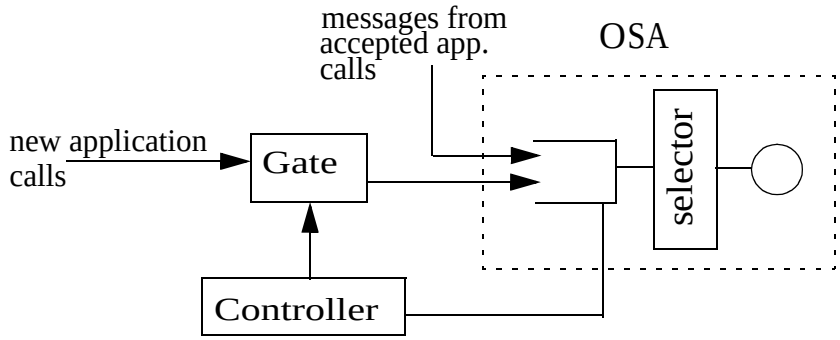


Figure 4. Model of the overload control mechanism

measurement data and determines what action that has to be taken by the *gate*, which regulates the acceptance of new application calls.

The objective of the overload control scheme is to keep all time constraints for the accepted application calls. As different applications can have different time constraints the *selector* has to decide in which order the messages in the gateway should be served. The selector uses an Earliest Deadline First (EDF) scheduling algorithm, see Lui et al [12]. The controller performs measurements of the load status in the gateway to check if it is possible for the selector to pick the messages in such order that none of the time constraints is expired. If not, the controller orders the gate to decrease the acceptance of new application calls. If the time constraints can be kept the gate is told to increase the acceptance of new application calls if possible.

5.1 Gate

When the gateway is overloaded, the gate starts to reject application calls. Since each application is guaranteed a minimum rate of accepted application calls per second, we have chosen to let the gate use a call gapping method, see Berger [13], to reject application calls. The time is divided into small intervals of a certain length, and then the first application call in the interval is accepted. The interval lengths depend on the guaranteed rate the application has. If, for example, an application is guaranteed at least 10 calls per second, this corresponds to a time interval of 0.1 seconds. During an overload situation (load level 1) the gate introduces call gapping on the lowest priority applications. If this action is not enough and the overload condition remains after *X* seconds, call

gapping is introduced on application calls of the next priority level, and so on every *X*:th second until applications from all priority levels have their calls rejected according to the call gapping method. The parameter *X* should be set taking into account the capacity of the gateway. If a severe overload condition (load level 2) appears all the priority levels are blocked at once, only letting the guaranteed amount of application calls through.

5.2 Controller

For each arriving or departing message the controller checks if the time constraints for the messages waiting in the queue may fail if the message is admitted. The following condition should of course always be fulfilled:

$$N(T_k) \cdot x_{GW} \leq T_k, \quad (1 \leq k \leq N) \qquad \textbf{(EQ 3)}$$

If not fulfilled, application calls with time constraint T_k will most probably fail even if the gate starts to reject arriving application calls at this stage.

Figure 5 can be used to easier explain the calculations that is performed by the controller to decide the current load condition in the Gateway. Each time constraint can be seen as an insertion point in the waiting queue for an application of a certain time constraint. When a message is inserted into the queue, this can be seen as some kind of time axis where the messages proceeds along this time axis as the distance to their deadline. When the execution of one message is completed the next message at the front of the queue is executed.

While a message with time constraint T_3 is waiting to get first in the queue it is possible for new messages with time constraint T_1 and T_2 to arrive at the queue with a closer deadline and thereby get a closer position to service. This means that during the interval of length $T_3 - T_2$ after the arrival of a time constraint 3 message, all applications with time constraint T_1 or T_2 will have a closer deadline. Then during another interval of $T_2 - T_1$, arriving messages with time constraint T_1 will have a closer deadline. Therefore, the condition in Equation 3 can be improved to also include the guaranteed rate of new application calls. This condition can be expressed as

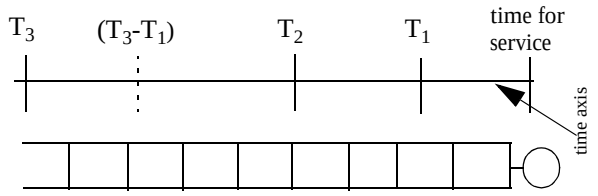


Figure 5. Abstraction of controller mechanism

$$\left(N(T_k) + \left(\sum_{j=1}^{k-1} \lambda_j \cdot (T_k - T_j) \right) \right) \cdot x_{GW} < T_k \qquad 1 \le k \le N \qquad \textbf{(EQ 4)}$$

If this condition is fulfilled and if the gate only let the guaranteed application calls through, the messages currently in the queue will most probably be served within their time constraints.

However, the controller should also check that the condition in (4) not is violated in the future by admitting too many calls from applications with less tough time constraints. Assume for example that an application call with time constraint T_3 arrives at time t_0. Let A be the set of all calls with a deadline in the interval $[t_0 - T_1, t_0]$ at this arrival. After $T_3 - T_1$ seconds all the calls in A will have a deadline that is less than T_1 seconds in the future. But in the time interval $[t_0, t_0 + T_3 - T_1]$ messages with time constraint T_2 and T_1 may have arrived to A and these messages will also have a deadline that is less than T_1 seconds in the future.

If a burst of application calls with time constraint T_i arrives and we start to reject calls from all priority levels, then the maximal number of messages that might be additional to an interval of length T_k, until all deadlines remain less than T_k, can be expressed as

$$\sum_{j=k}^{i-1} \lambda_j \cdot T + \sum_{j=1}^{k-1} \lambda_j \cdot T_j \qquad \begin{matrix} 2 \le i \le N \\ i > k \end{matrix}$$

$$T = \begin{cases} T_k & T_k < (T_i - T_j) \\ (T_i - T_j) & T_k \ge (T_i - T_j) \end{cases} \qquad \textbf{(EQ 5)}$$

and the execution time of these should be added to the execution time of all initial messages in the interval of length T_k. Just as before we also have

to include that new application calls might arrive during the execution of the messages in the interval. This new condition can be described as

$$\left(N(T_i) - N(T_i - T_k) + \left(\sum_{j=1}^{k-1} \lambda_j \cdot (T_k - T_j) \right) + \sum_{j=k}^{i-1} \lambda_j \cdot T + \sum_{j=1}^{k-1} \lambda_j \cdot T_j \right) \cdot x_{GW} < T_k \quad \begin{matrix} 2 \le i \le N \\ i > k \end{matrix} \quad \textbf{(EQ 6)}$$

where T is the same as in Equation 5. This constraint has to be fulfilled for all possible combinations of T_k and T_i, where $i > k$.

If conditions (4) and (6) are fulfilled, the controller decides that the system has a high probability to succeed without too many expired deadlines. If any of the two conditions fail, the controller signals overload to the gate. Too further decrease the number of expired deadlines and to get a more calmly behaviour, the controller uses a marginal when signalling for overload. This marginal is created by multiplying the right hand of the conditions with a marginal factor, $f<1$. If any of the conditions are violated when the right hand side is multiplied with f, the controller signals overload (load level 1). If any of the conditions are violated without the marginal factor, the controller signals severe overload (load level 2) to the gate.

6. Simulation Parameters

In the simulations we have used 10 applications with different behaviour. In our implementation we support two different time constraints and three priorities for the applications. The different applications also differ in execution times in the OSA Gateway, delays in the AS, delays in the network and the number of times an application needs to pass the gateway. Table 1 shows the configuration of the applications used in our simulations. The sequence diagram of application 1, 2 and 3 is the same as the example shown in Figure 2. The sequence diagrams of the other applications are not shown, but their most important properties can be seen in the table. The reason for that we have one exponentially distributed and one deterministic service time in the network for application 1, 2 and 3 is that these correspond to some kind of call establishing look alike applications. Such applications have to execute twice in the network. The first time is when the call is connected to the callers phone. In this case a deterministic delay is used, since there is probably some kind of auto phone pick up function for the caller. The

Table 1. The configuration of the different applications.

App.	Execution times in the OSA Gateway	Execution times in the AS	Execution times in the network	Priority	Time constraint
1	0.001, 0.002, 0.002, 0.002, 0.002, 0.001	0.001, 0.001, 0.001	0.008, exp(2,0)	1	0.1
2	0.002, 0.003, 0.003, 0.003, 0.003, 0.002	0.002, 0.002, 0.002	0.01, exp(0.01)	2	1.0
3	0.0001, 0.0002, 0.0002, 0.0002, 0.0002, 0.0001	0.001, 0.001, 0.001	0.01, exp(2.0)	3	0.1
4	0.0015, 0.0025, 0.0025, 0.0015	0.0015, 0.0015	0.0125	1	1.0
5	0.0001, 0.0002, 0.0002, 0.0001	0.0001, 0.0001	0.0008	2	0.1
6	0.001, 0.002, 0.002, 0.001	0.001, 0.001	0.008	3	1.0
7	0.002, 0.002	0.001, 0.001	0.018	1	0.1
8	0.003, 0.003	0.005, 0.005	0.008	2	1.0
9	0.0002, 0.0002	0.0001, 0.0001	0.0008	3	0.1
10	0.0015, 0.0015	0.0001, 0.0001	0.0035	3	1.0

second execution correspond to the B party pick up time. This pick up time is modelled to be exponentially distributed.

Priorities 1, 2, and 3 correspond to guaranteed rates, d_l, of 10, 5, and 0.5 accepted calls per second, respectively.

The marginal factor *f*, used in the controller, is set to 0.9. To prevent the system from oscillating, the load level is only changed if the controller detects overload for five consecutive arrivals or departures. The parameter *X* introduced in Section 5.1 is in our simulations set to 50ms.

During the simulations we have used different values of x_{GW} depending of which features that are prioritized. We think that two different values can be motivated. Either we use the upper quartile of the execution times in the Gateway for all messages from all applications or we use the largest execution time in the Gateway. The choices are referred to as configuration 1 and configuration 2. The choice of x_{GW} to the upper quartile is motivated as it seems not realistic that only applications using messages with the largest execution times in the Gateway are used. Thereby this will give a good estimation of the maximum mean of the execution time for the messages in the queue, but with this choice we

cannot guarantee the rate of expired application calls. The choice of x_{GW} to be the largest execution time among all applications is motivated as this will result in a very limited amount of expired application calls. This reasoning counts for all conditions where x_{GW} takes part in this article

7. Results and Discussion

In this section the overload control mechanism will be evaluated. Simulation results are presented and the gain of the proposed overload control mechanism is discussed.

We first consider arrivals according to a poisson process where the rate, λ, is changed every 25:th second. At first λ is 10 calls/s, and is then increased to 20, 25 and 50 calls/s. In Figure 6 the resulting rates of completed application calls per second for each of the three priorities are shown when configuration 1 is used. During the first 25 seconds no calls are rejected, since the total number of arriving application calls is below the capacity of the Gateway. However, after 25 seconds λ is almost equal to the capacity of the Gateway and the priority 3 application calls are slightly affected. After 50 seconds, the total arrival rate exceeds the capacity of the Gateway and, therefore, application calls with the lowest priority are rejected. After 75 seconds, calls from all priority levels are rejected, but none of the priority levels have all their application calls rejected. This is because of the guaranteed amount of application calls. In the realization in Figure 6 about 5% of the served application calls were so-called expired calls. However, most of the expired application calls break their time constraints only with a few percent of their constraints.

If configuration 2 is used the outcome of a typical realization when steady state is used is seen in Figure 7. In this realization 0% of the served application calls are expired calls.

It is interesting to see what can be gained with a priority based rejection mechanism as proposed. An estimation of how good the outcome is can be performed by using the utility measure. Assume that the applications of priority 1, priority 2 and priority 3 corresponds to utility 3, 2 and 1 respectively. Then we can get a measure of the gain by calculating how much time the processor has spent on the different applications and take into account the utility of each application. The revenue of a realization can then be defined as

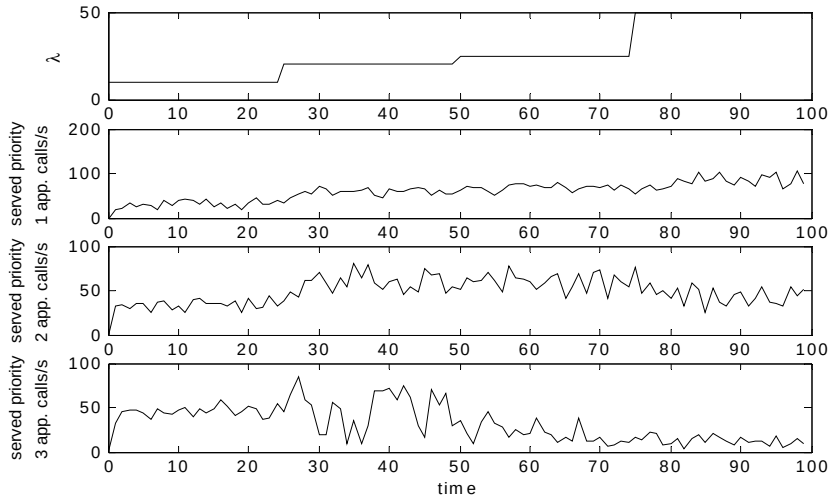


Figure 6. The outcome of a realization when configuration 1 is used. The top graph shows the mean λ of each application. There are three priority 1 applications, three priority 2 applications, and four priority 3 applications

$$\sum_{l=1}^{R} s(l) \cdot x_{tot}(l) \cdot U(l) \qquad \textbf{(EQ 7)}$$

where *s(l)* is the number of served application calls from application *l* during one simulation.

In the realization shown in Figure 6 we get a total revenue of 213 units. The same calculation for the gain of the realization shown in Figure 7 results in a total revenue of 214 units. To be able to do a comparison of this result we have done the same simulation where we have used an ordinary random rejection method where 50% of the application calls are rejected during an overload (load level 1) situation, and where all application calls except the guaranteed amount is rejected during an severe overload (load level 2) situation. When this overload control mechanism is used we get a total revenue of 199 units if the parameters are set so that 3,5% of the served application calls are expired. Observe that when this mechanism is used we cannot guarantee that the

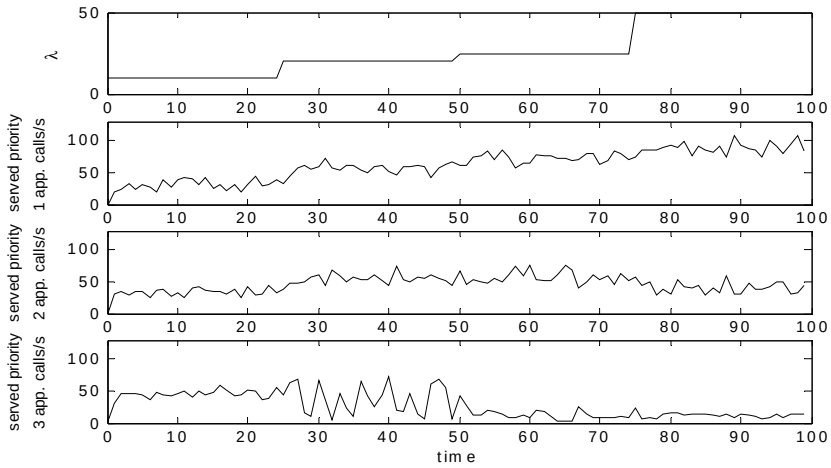


Figure 7. The outcome of a realization when configuration 2 is used. The top graph shows the mean λ of each application. There are three priority 1 applications, three priority 2 applications and four priority 3 applications

requirement of the guaranteed rates of accepted application calls per second is fulfilled during an overload (load level 1) situation.

In the plots it is seen how configuration 2 rejects more application calls than configuration 1 during an overload situation. It can also be discerned that there are larger differences between the number of accepted calls from different priorities when configuration 2 is used. When the random rejection mechanism is used all applications have about the same amount of application calls served per second. As a consequence of this we get less revenue.

However, arrivals according to a Poisson Process is probably not so realistic. Therefore we have also used an MMPP with six states to generate arrivals. The states correspond to arrival rates 0, 10, 20, 30, 40 and 50 calls per second. The mean time in a state is two seconds and when we leave a state, we enter one of the others with equal probability. Under these circumstances and when configuration 1 is used the rate of expired calls is about 0.15%. And when configuration 2 is used the rate of expired calls is 0%. As we assume that the mean execution time is higher in configuration 2 than in configuration 1 we cannot allow the same queue length. Thereby the queue will become empty more often as a consequence if it is filled by messages with short execution time, and thereby the utilization is lower if configuration 2 is used.

If we do the same calculations of the gain as before, but with an process of the arrivals according to MMPP we get a total revenue of about 2350 units when configuration 1 is used during a simulation of 1000 seconds. The same calculation for a configuration 2 simulation during 1000 seconds results in a total revenue of 2280 units. If we use the earlier described random rejection mechanism, with the parameters set such that the rate of expired calls is 0.1%, we get a total revenue of 2180 units.

Clearly we get a better gain when the priority based overload control is used. The values of x_{GW} have different advantages. The choice of this value should be made depending of how the contract is written and what the arrival process look like. During an MMPP arrival process, which we think is the most realistic, the use of the upper quartile results in a better utilization and a higher gain than if we use the largest execution time, which is an argument for configuration 1. But we also get a higher fraction of broken deadlines, which is an argument for configuration 2.

8. Conclusions

We have modelled an overload control mechanism for an OSA gateway. The Overload control mechanism that is proposed is designed to support two probable requirements in an OSA architecture. It is able to guarantee a minimum rate of accepted application calls per second dependent of which priority an application correspond to. It is also designed to make sure that application calls that are accepted will meet their time constraint with a high probability. Further, discussions are held and a proposal is presented of how the priorities can be set in order to maximize the revenue for the owner of the Gateway.

The different parts in the overload control mechanism are build independent of each other such that either the gate, the selector or the controller can be exchanged without any affect on the other parts. Also the utility function can be exchanged.

By simulations the overload control is evaluated and the appearance of the wanted behaviour is verified. Also, we have shown that the total gain of the served application calls is higher when using the priority based rejection mechanism compared with a random rejection mechanism.

Acknowledgement

This work has partially been financed by the Swedish Research Council, contract no 621-2001-3053.

References

[1] The 3GPP home page, "www.3gpp.org"

[2] R. Rajagopulan, “The impact of Open Service Access on Network Services”, http://www.wmrc.com/businessbriefing/pdf/wireless_2003/Technology/lucent.pdf, 2002.

[3] L. J. Forys, “Performance Analysis of a New Overload Strategy”, ITC 10, 1983

[4] X. H. Pham, R. Betts, “Congestion Control for Intelligent Networks”, 1992 International Zurich Seminar on Digital Communications, 1992

[5] M. Kihl, C. Nyberg, “Investigation of overload control algorithms for SCPs in the intelligent network”, Communications IEE Proceedings, vol. 144, 1997

[6] R. Melen, C. Moiso, S. Tognon, “Performance evaluation of an Parlay gateway”, http://exp.telecomitalialab.com/pdf/06-MOISO4.pdf, 2001

[7] J. Andersson, M. Kihl, C. Nyberg, “Performance analysis and modelling of an OSA gateway”, In the proceedings of PWC2003, 2003

[8] R. M. Stretch, “The OSA API and other related issues”, B T Technol J., Vol. 19 No 1, 2001

[9] ETSI standard 201 915-4 v1.3.1, “Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control SCF”, 2002

[10] ETSI standard 201 915-3 v1.3.1, “Open Service Access (OSA); Application Programming Interface (API); Part 3: Framework”, 2002

[11] Eurescom Technical Information, ”Parlay/OSA Business Models: An Operator’s Perspective”, December 2002

[12] C. Liu, J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”, Journal of the Association for Computing Machinery, Vol. 20 No. 1, 1973.

[13] A. Berger, “Comparison of Call gapping and Percent blocking for overload control in distributed switching systems and telecommunications networks”, IEEE Transactions on Communications, vol. 39, 1991

PAPER IV

Overload Control of a Parlay X Application Server

Jens Andersson[1], *Maria Kihl*[1] *and Daniel Söbirk*[2]

[1]Department of Communication Systems
Lund Institute Of Technology, Sweden
Box 118 221 00 Lund
E-mail: {jens.andersson, maria.kihl}@telecom.lth.se

[2]Appium AB
daniel.sobirk@appium.com

IV

Abstract

Parlay/OSA is a service architecture developed to increase the number of potential application developers of telecommunication services. By abstracting the telecommunication network resources into building blocks, it is possible to create new applications without any deep knowledge of telecommunications. To be able to reach even more developers, Parlay X was introduced as an extension to Parlay/OSA to provide applications that reside in the Internet the ability to use network resources from a telecommunication network via so-called web services. An application residing in the Internet can reach a network capability via an entity called an application server, which converts the building blocks to appropriate communication with the network. In this paper an application server is modelled, its performance is investigated and a load control mechanism is proposed. The load control mechanism is designed to support constraints of different kinds as Parlay X is a contract driven architecture. Simulations are used to evaluate the behaviour of the load control mechanism.

Keywords: Application Server, overload control, load balancing, admission control, service architecture, performance evaluation

1. Introduction

In the telecommunication networks used today, each network operator have their own service architecture. The service architecture is built on the Intelligent Network (IN) technology [1]. Since the IN technology was introduced it has been easier to create and introduce new services and applications. But thorough knowledge and skills regarding the telecommunication signalling and protocols has been a requirement for a service creator. The limitations of the telecommunication networks have been highlighted by the explosive growth of the Internet. One of the main reason of the growth is the large number of software developers. In the Internet any software developer has the capability to create an application reachable for any Internet user, a fact that has inspired the telecommunication operators to think in new directions.

The first serious initiative of a service architecture that makes the creation of new applications less complicated was Telecommunications Information Networking Architecture (TINA) [2]. The main idea with TINA was to provide developers with building blocks corresponding to different telecom features. This initiative remains at the research level, and no TINA architecture is implemented and used today.

However, the TINA initiative has lead to several new groups that has tried to challenge complexity of creating new applications. Parlay [3] and JAIN [4] are examples of groups that are developing Application Program Interfaces (APIs) that allow applications to access network functionality. In this way the telecom networks open up to IT developers, and development of new and innovative applications is foreseen. As a complement to the Parlay APIs, another group started to develop some APIs that could be used to provide access to applications on top of a Universal Mobile Telecommunications System (UMTS) network. The group developing these APIs is called Open Service Access (OSA) and is part of the 3GPP [5]. The Parlay/OSA APIs are standardized, both by ETSI [6] and the 3GPP.

To reach even more IT developers the so called Parlay X web services have been introduced, developed by the Parlay X working group within Parlay. Parlay X involves more abstraction of the building blocks of telecom capabilities and with this architecture it is possible for applications that reside in the Internet to reach a telecom resource by a single command via a so called Application Server (AS).

An AS is connected to the Parlay/OSA environment and thereby acts as a gateway between the applications and the network from the applications point of view. The AS is a distributed system that performs the mapping between Parlay X commands and Parlay/OSA commands. That the AS is distributed means that it contains different processing nodes, which may be involved during the service of a request.

In the context of service architectures a common problem is overload. The congestion we are interested of in this article occurs when there are too many applications that try to make use of the same AS at the same time. To avoid congestion, load balancing algorithms have to be combined with admission control. The strategy of load balancing is to spread the load among the nodes. The admission control strategy is to reject some of the requests if there is not enough capacity to serve all of them in a sufficient way.

The load balancing and admission control methods are well known concepts for congestion avoidance. In Kihl et al. [7] load balancing algorithms for a TINA network are discussed. Berger [8] has a discussion about different admission control methods and Pham et al. [9] gives an example of an overload control mechanism for IN. Dahlin [10] discusses what impact old information about the load status should have on the decisions. An article that treats a distributed architecture is Bakshi et al. [11].

This paper gives an overall description of the Parlay/OSA and Parlay X service architecture. Extra attention is paid to describe the Application Server and its functionality. We will also present an overload control algorithm that is designed for a distributed Application Server with messages of different priorities, which need different treatment. The algorithm will also consider delayed information. Investigation of the feasibility to minimize the constraints of hardware in the load balancing / admission control entities is discussed. The performance of the algorithms and methods is investigated with simulations.

In the next section a description of some overload control methods is given. A description of a Parlay/OSA and Parlay X environment is presented in the section after that. The application server in a Parlay X architecture is then described. The objectives of this paper are presented in the section after that, followed by the description of how we modelled the AS and its surroundings. Then the load balancing algorithms and the admission controlling algorithms are described. Simulation parameters

are followed by results and discussions. Finally some conclusions are presented in the last section.

2. Load Balancing and Admission Control Methods

In this section the basics of load balancing and admission control are explained. Some extra attention is paid to specific methods that are used later in this paper.

2.1 Load Balancing

The purpose of a load balancing algorithm is to distribute the load in such way that all processing nodes in a distributed environment is equally loaded. In a distributed architecture with full information of the current load (e.g. non-delayed information), the load balancing is trivial, but in real systems with multiple traffic sources and where delays occur load balancing is not trivial. Many complex algorithms have been developed with different advantages.

However, very often it is shown that simple algorithms like Round Robin (RR), see Stallings [12], is almost as good as much more advanced and specialized algorithms. The method of RR is to remember how much traffic that has been sent to a specific node and then let all nodes in the system get the same amount. If the different nodes in a system have different capacities the traffic flow should be weighted to the different nodes. The algorithm is now called Weighted Round Robin (WRR). If we denote the amount of the traffic that node *i* with capacity C_i should get as A_i, then

$$A_i = \frac{C_i}{\sum_j C_j} \qquad \text{(EQ 1)}$$

can be used to calculate the amount of traffic.

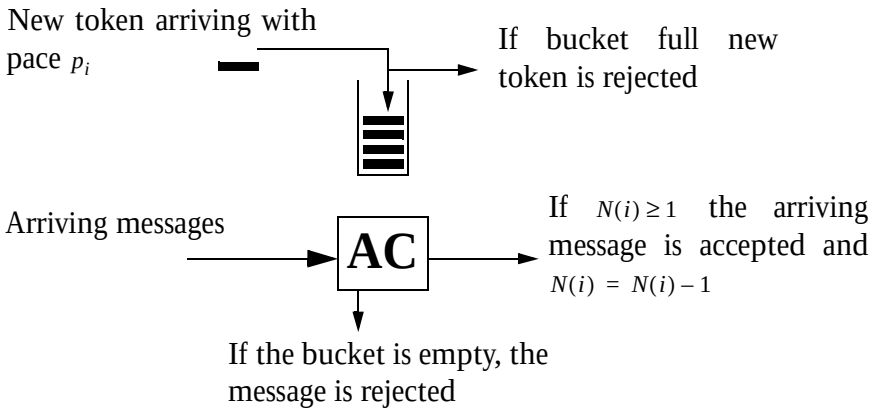


Figure 1. Description of a token bucket admission controller

2.2 Admission Control

The purpose of admission control is to protect the system when there is not enough capacity to serve all requests for service. Therefore some of the requests have to be rejected, and this can be done according to different algorithms. Example of an admission control algorithm is the token bucket. A token bucket regulator has a flow of p tokens per second to bucket of size l. The number of tokens in the bucket is denoted N. If $N = l$ the arriving tokens to the bucket are rejected. When a request arrives we check if $N \geq 1$. If this is the case N is decreased by 1 and the message is accepted, otherwise it is rejected. Figure 1 describes an admission controller with a token bucket.

3. Description of a Parlay/OSA and Parlay X Environment

The main advantage of introducing a Parlay/OSA environment is the increased ease of creating new applications that may use the resources of a telecommunication network. This is also the main reason why Parlay/OSA has been developed. Example of a network resource can be setting

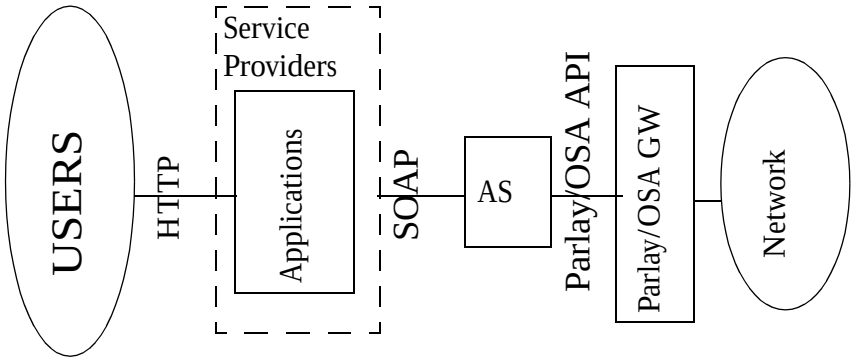


Figure 2. Architectural picture of an AS and Parlay/OSA environment

up a call. Earlier, a developer was required to have great technical skills and deep knowledge of telecommunication protocols to be able to implement an application in a service architecture. With Parlay/OSA the network capabilities are abstracted and reached by APIs. The API introduces so called Service Capability Features (SCFs) and each resource provided by the network is abstracted to an SCF. So in order to use a certain network capability, an appropriate SCF has to be called. More details about the SCFs can be found in [13].

3.1 The Architecture

The service architecture can be applied to any telecommunication network, and it is not specified in which network the users should reside to be able to use the architecture. One of the most discussed and promising topologies is when the end user of an application is connected to the Internet. This is the architecture that we will treat in this article. A Parlay/OSA architecture connected to the Internet is often referred to as a Parlay X architecture where the network resources are abstracted to so called Parlay X Web Services, see [14].

The architecture of a typical Parlay/OSA environment connected to the Internet can be seen in Figure 2. The main elements needed to describe the architecture are Users, Service Providers (SPs), Application Server (AS), Parlay/OSA Gateway and a Network.

- Users: The case we investigate is when the users of the applications are connected to the Internet.

- SP: The SPs are hosting the applications. Either the SPs own the applications or the owner of an application hire disk space at an SP. In a Parlay X architecture the SPs can provide the application developers the capability to make use of a network resource via a request using Simple Object Access Protocol (SOAP) transported by Hyper Text Transfer Protocol (HTTP).
- AS: An AS translates the SOAP requests to Parlay/OSA commands. This means that the AS is a gateway to a network resource from the SPs point of view. The AS is owned by the network operator.
- Parlay/OSA gateway: The Parlay/OSA gateway is owned by the network operator and handles all advanced non-abstracted communication with the network.
- Network: The network can be either a mobile or a fixed telecommunication network. Any network that gains from increased ease of creating new applications can adopt the Parlay/OSA service architecture.

3.2 Communication from Application to Network

A request for a network resource of any kind is denoted as a message. A typical request for service starts with some HTTP communication between an end user and an application (for example click to dial). SOAP is used to specify what kind of network resource that the application aims to use. The SOAP message is sent to the AS where the messages are converted to call the appropriate SCF in the Parlay/OSA gateway. The processing that really takes place in and with the AS is described in a later section. A SOAP message arriving at the AS can cause much communication between the AS and the gateway. The Parlay/OSA gateway is directly connected with the network. The SCFs are specified by Parlay/OSA standards, but it depends on the underlying network what SCFs that are supported. More about the Parlay/OSA gateway can be read in [15].

To be able to distinguish between different messages, each message has a call-id which is encapsulated to the SOAP level. As the SOAP messages are encapsulated by HTTP it is not feasible to see what call-id a message has without unwrapping, but HTTP supports the ability to see what kind of SOAP request that is encapsulated.

Parlay/OSA and Parlay X is not yet widely used. The operators who has shown the largest interest so far are operators for the fixed networks. They see great opportunities in providing application initiated calls. Applications providing such features would make it feasible to establish conference calls at a certain time where the initiator has the control to shut down the conference at any time, or after a predefined time. The ease of use and the lucidity of the applications will also increase when they are presented on a monitor and controlled by text input.

3.3 Contracts

Parlay/OSA is a contract driven architecture. There are contracts between the AS and the Parlay/OSA gateway and between the AS and the SPs. The contracts include several constraints and restrictions, but we are only interested in the variables concerning the performance. Each SP that makes use of an AS must have a contract. Relevant facts included is how many application calls from a certain SP that at least should be accepted each time unit, and also a time constraint for the maximal delay. A maximal number of application calls per time unit from an SP will also be agreed. If an SP does not fulfil the constraints this will lead to that the Load Balancer/Admission Controller (LB/AC) do not have to fulfil its undertakings.

4. The Application Server

A detailed view of an AS is shown in Figure 3. An AS is a distributed architecture where several Parlay X to Parlay converters (PX-P converters) are active at the same time. There is also a coexistence of several SPs, each one with their own contract and constraints that should be fulfilled. To be able to fulfil the constraints and avoid congestion at the PX-P converters a Load Balancing / Admission Control (LB/AC) mechanism is used. It is important to distinguish between the LB and the AC. The aim with the load balancing is to distribute the messages among the PX-P converters such that about the same load is reached at each PX-P converter. The aim with the admission control part is to reject some messages when the PX-P converters are overloaded. When an application sends a message to the AS it is received by the LB/AC. The LB/AC decides whether the message should be rejected or accepted and to which PX-P converter it should be

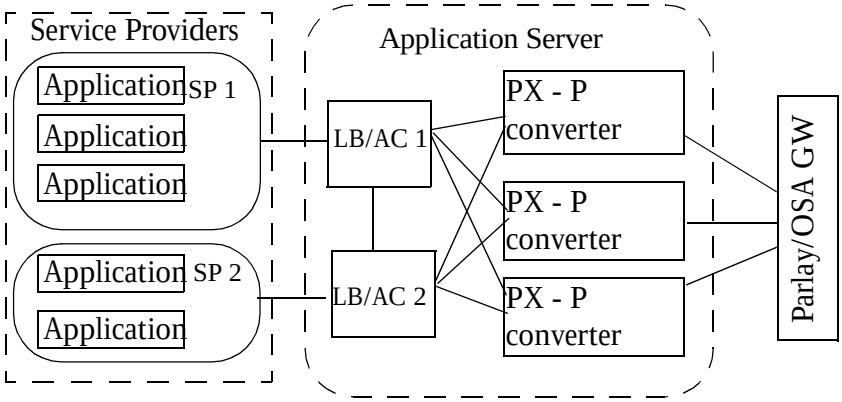


Figure 3. Detailed view of the AS

forwarded if accepted. In the PX-P converter the message is mapped to corresponding Parlay/OSA communication, to serve the SOAP message request. The architecture of the PX-P converters is built on a Common Object Request Broker Architecture (CORBA) platform. Internal communication between the PX-P converters is possible using CORBA messages. An AS typically consists of 2 to 30 PX-P converters. It is difficult to predict how many SPs an Parlay/OSA architecture will embrace, as it is dependent of the size of each SP and the popularity of the applications.

The case we investigate is when there are messages of three different kinds and with different priorities supported in the AS. These are

- EndCall (priority 1), ends the ongoing call
- GetCallInfo (priority 2), requests information about the connected parties etc.
- MakeACall (priority 3), request to create a new call

This is a scenario investigated by demand of the industry. It is interesting that the MakeACall messages have a lower priority than the others, because the contracts only concerns the rate of accepted MakeACall messages. The explanation is simply that all accepted calls should be taken cared of. We will later on investigate and discuss the ratio of the different messages.

In the considered architecture the application calls are session based. This means that the same PX-P converter must take care of the GetCallInfo messages and EndCall message corresponding to the same

application call. Therefore the call-id is the same for all messages belonging to the same call. The call-id for an application call is assigned by the PX-P converter at the arrival of a MakeACall request. Each PX-P converter is dedicated an interval of call-ids to choose among. If the call-id is accessible in the LB/AC part, a list is maintained over which call-ids that should be served by which PX-P converter. The calls can also be ended from the network side (e.g. the parties hang up), without noticing the AS.

4.1 Load Control Mechanism

The load control mechanism can be described by Figure 4. The LB/AC consists of a gate/controller and a monitor. The monitor performs the measurements of the load at the PX-P converters. Each time a message (request) is sent from the LB/AC to any of the PX-P converters a thread is used. The same thread is later used for the response of the request, see next section. Exactly one thread is used for each request and therefore it is feasible to measure the time between request and response to get an estimate of the current load. It must be observed that the measured time does not give a precise measure of the current load condition. Instead we get information about the load when the served message was sent. It is in the gate/controller the algorithms for the load balancing and admission control is implemented. Based on the information the gate/controller gets from the monitor it decides whether or not the gate/controller should reject a new message of a certain kind. More about the algorithms implemented in the gate/controller in a later section.

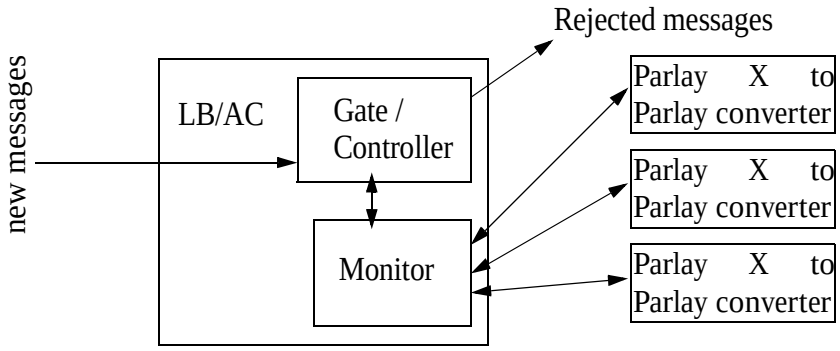
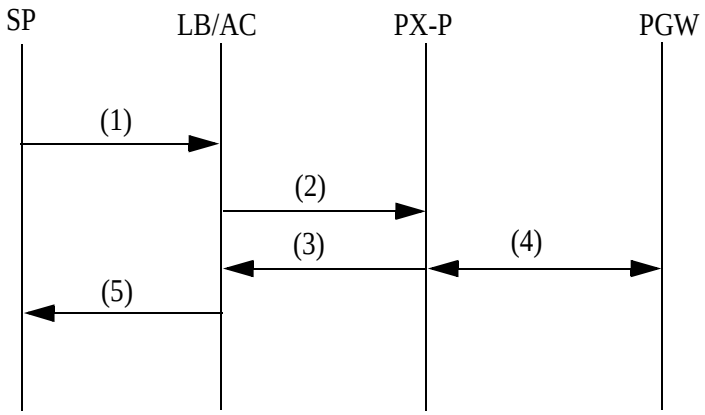


Figure 4. The load control mechanism



SP: Service Provider
LB/AC: Load Balancer / Admission Control
PX-P: Parlay X to Parlay converter
PGW: Parlay Gateway

Figure 5. Sequence diagram of MakeCall

4.2 Example of a MakeACall Request

In Figure 5 the sequence diagram for a MakeACall message is shown. The first message (1) is a makeACall.

When the LB/AC receives the MakeACall message it decides whether to accept or reject the new call. If the call is accepted a MakeACall message (2) is sent to PX-P converter. The message is unwrapped and the converter sends a message (3) to inform the LB/AC that the request now is processed. The LB/AC forwards the message (5) to the SP. When this message is received the SP might at any time send GetCallInfo and EndCall messages with the same call-id. The processing in the PGW involves different amount of communication dependent of how many participants the SOAP message attend to connect by the MakeACall message. The communication with the PGW consist of calling appropriate Service Capability Features (SCFs).

The sequence diagram for a GetCallInfo message is almost the same as shown in Figure 5 excluding message (4). An EndCall message has a sequence diagram identical to Figure 5.

5. Objectives of This Paper

The objectives are to investigate how different overload control algorithms can be implemented to maximize the throughput and to get a robust system at the same time as the constraints are fulfilled. Another important issue is to investigate how the constraints of hardware for the LB/AC part can be minimized. The most time consuming job is wrapping and unwrapping of the SOAP requests and therefore it is interesting from the hardware point of view to investigate whether it is feasible to avoid unwrapping the SOAP requests in the LB/AC part. If the SOAP messages are not unwrapped it is not feasible to reach the call-id and thereby there will be no information about which PX-P converter that should treat the GetCallInfo and EndCall messages.

6. Model of a Distributed Application Server System

The modelled AS system consists of n SPs and m PX-P converters where converter i has the capacity C_i. Each SP might include several applications but as the contracts are agreed between the SPs and the AS, the number of applications is not important. As each SP is connected to their own LB/AC, see Figure 3, a single SP is the traffic generator to the LB/AC and thereby the AS. There are reasons to believe that the traffic to ASs will be rather bursty, since the ASs can be reached from the Internet. It is actually the arrival process at the web servers hosting the applications, which consequently will be the arrival process of new service calls for the AS. In the context of web servers the arrival process is often modelled as a Markov Modulated Poisson Process (MMPP), see Chen et al. [16]. We believe that MMPP is a good assumption also in the context of ASs.

Our model can bee seen in Figure 6. As we investigate the problem of overload in the PX-P converters we assume that there is no overload at the SPs or in the Parlay/OSA GW. Therefore the SPs and the Parlay/OSA GW are modelled as delays, as there is assumed to be enough capacity for instant service. The LB/ACs are modelled as single server FIFO queues. The service times in the LB/AC are assumed to be deterministic as the same decisions and processing should be performed independent of message. We denote the service times $\bar{x}_{wrapped}$ or $\bar{x}_{unwrapped}$ dependent on

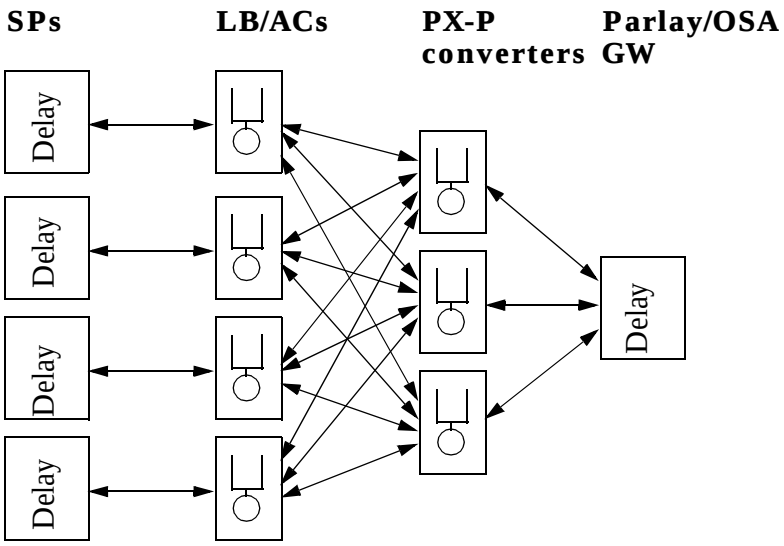


Figure 6. The model of an AS

if the SOAP messages are unwrapped or not. The PX-P converters are also modelled as single server FIFO queues. In the case when the LB/ACs do not unwrap the SOAP messages it may occur that the GetCallInfo and EndCall messages are sent to the wrong PX-P converter. As the application calls are session based the wrong PX-P converter cannot complete the requests without consulting the converter handling messages with the specific call-id. Therefore CORBA messages are used for this purpose. The processing time for handling a SOAP message in a PX-P converter depends on the capacity and is denoted $\bar{t}_i$ for converter i. The processing time for handling a CORBA message is $\bar{t}_i/4$.

We assign CORBA messages higher priority than the SOAP messages in the converter queues, since the messages sent to wrong converter otherwise must wait in the queues too long.

7. Overload Control

In this section we will propose a Load Balancing and Admission control mechanism for robust protection of an AS. Two cases will be considered, the unwrapped and the wrapped SOAP case. Depending on which case

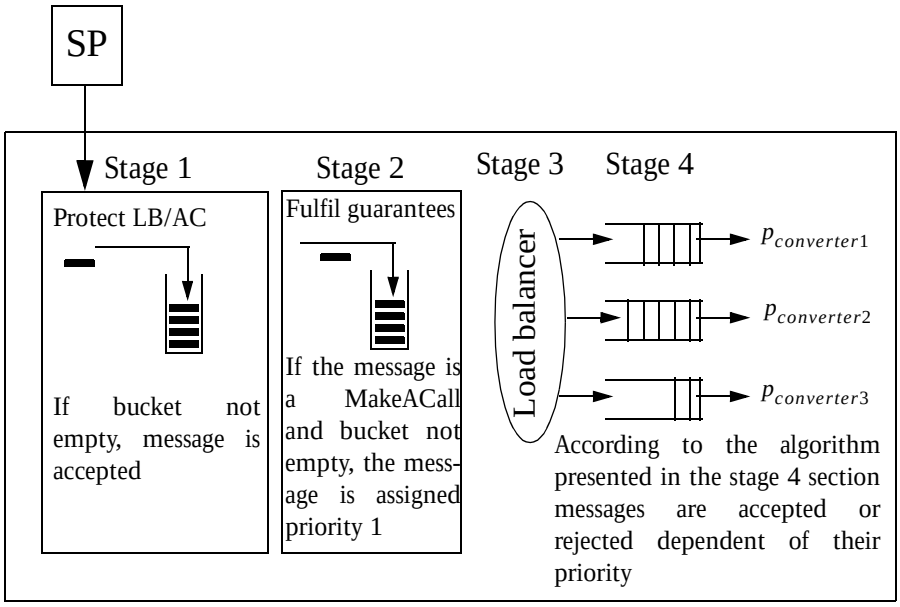


Figure 7. Overview of the different stages in the LB/AC

that is considered the call-id can be used or not. We define overload as when the waiting times for the users are too long. The waiting time for a user is defined as the time between message (1) and (5) in Figure 5. The time constraint for the applications is denoted τ. The load control mechanism must also consider that each SP *k* has a guaranteed rate of d_k MakeACall messages per second. To explain the complete load control mechanism, four stages are used, see Figure 7.

7.1 Rough Admission Control (Stage 1)

To protect the LB/AC node from overload a rough admission controller is used. The stage 1 mechanism rejects messages without treating them if the arrival rate is too high. In the contracts a maximal number of messages sent per second, denoted r_i will be agreed for PX-P converter *i*. If more than r_i messages are sent during a second the LB/AC does not have to fulfil the constraints against the SP. We propose that the value of r_i is chosen such that

$$r_i \cdot 2 \cdot \bar{x}_{(un)wrapped} < 1 \quad \textbf{(EQ 2)}$$

to be able to serve all accepted messages and their potential responses returned to the LB/AC. To control that the number of messages do not exceed r_i messages per second we propose the use of a token bucket of size r_i, with pace r_i. If there is no token when a message arrives it is rejected without informing the SP.

7.2 Constraint Control (Stage 2)

When the LB/AC has explored which category an arriving message belongs to, the LB/AC must control that the constraint of d_k accepted MakeACall messages per second is fulfilled. To fulfil the guaranteed number of accepted MakeACall messages for SP *k* we use a token bucket of size d_k and tokens arriving with rate d_k. If there is a token in the bucket when a MakeACall message arrives the message is given priority 1, the same priority as the EndCall messages, and will always be accepted at later stages. Notice that each LB/AC only have one bucket for controlling the guaranteed rate of MakeACall messages. If the bucket is empty when a MakeACall message arrives, it is just forwarded to the next stage with the original priority, 3. Also the GetCallInfo and EndCall messages are forwarded with their original priorities 2 and 1.

7.3 Load Balance (Stage 3)

We propose the use of weighted round robin algorithm for load balancing, since it is known to be robust. The amount of messages a PX-P converter receives should be weighted by the capacity of the PX-P converters according to Equation 1. However, the algorithm should not be adopted to all messages (i.e. all messages should not be load balanced), as a consequence of the session based nature. In the case when the SOAP messages are unwrapped we have a feasibility to distinguish which message that correspond to which session. Therefore the wrapped and the unwrapped case result in different actions.

Unwrapped Case

Since the application calls are session based only the MakeACall messages are distributed with the round robin algorithm. The GetCallInfo and EndCall messages can only be served by a specific PX-P converter. Therefore it is not optimal to send a message to the wrong PX-P converter as this would result in extra processing and waste of total processor capacity. In the unwrapped case the information about which call-id that should be served by which PX-P converter is maintained in a table. There is no information on how many active application calls there are at the moment since the LB/AC is not noticed when a session ends from the network side. An assumption is that each application call correspond to about the same utilization of the capacity.

Wrapped Case

Just as in the unwrapped case the MakeACall messages will be distributed by the round robin algorithm. The call-ids are not accessible in the wrapped case so it is unknown which PX-P converter the GetCallInfo and the EndCall messages should be sent to. Therefore the round robin algorithm is used to distribute these messages as well. This is not the most effective algorithm from the hardware in converters point of view as they will have to process extra CORBA messages for $(m-1)/m$ percent of the GetCallInfo and the EndCall messages.

7.4 Admission Control (Stage 4)

During overload situations in the converters some kind of action must be taken. The admission control mechanism should choose which messages to serve and which to reject. Overload is defined as when the messages cannot be served within the time constraint. If a message is not served within the time constraint it is said to be an expired message. Notice that the measurements are performed of the time from request to response in the LB/AC, denoted $\Delta t_{measured}$. However, an expired message is defined as when the duration between request and response at the SP, denoted Δt_{user}, is larger than τ. The aim for the controller is to keep

$$\Delta t_{user} < \tau \qquad \textbf{(EQ 3)}$$

In the context of admission control we do not distinguish between the cases if the SOAP messages are unwrapped or not. This can be made as it is feasible to distinguish between SOAP messages at the HTTP level.

Proposed admission control algorithm

If a message is accepted at the first stage the message is unpacked and sent to the load balancer. The load balancer decides which PX-P converter that should treat the message. Each LB/AC has a buffer queue for each PX-P converter. When a message is load balanced it is sent to the end of the buffer queue for a certain PX-P converter. Statistics are then maintained on how many messages of priority *i* that are present in a certain queue and this number is denoted $N(i)$. To regulate the load at the PX-P converters we choose to regulate on the pace that messages are sent from the queue to the converter. A higher load will result in a lower pace. To predict the load in the converters we use the measured times $\Delta t_{measured}$, and express the current load as a load level. We introduce a number of threshold values, th_k, which are used for comparisons with $\Delta t_{measured}$. Based on the measured times the LB/ACs maintains a load-table where the load level is set to *k* according to the formula

$$(th_k \leq \Delta t_{measured} \leq th_{k+1}) \quad \textbf{(EQ 4)}$$

$\Delta t_{measured}$ is actually expressing the load status when the message was sent to the converter, but it can be predicted that load level *k* is close to the real load condition. Each load level *k* then correspond to a certain pace p_k. To avoid the LB/ACs from sending too many messages during a short time interval, which could lead to that the load conditions in the PX-P converters change too fast, the values of the paces should be set in such way that

$$n \cdot th_{k+1} \cdot p_k < C_i \cdot th_{k+2} \quad \textbf{(EQ 5)}$$

C_i is the capacity of the converter *i* and th_{k+2} adopts a maximal delay, th_{max}, that should not be exceeded if the threshold does not exist. C_i should be expressed as how many messages that is served in mean per second during full utilization. With the pace for the largest *k*, p_{max}, it must be fulfilled that

$$n \cdot p_{max} < C_i \quad \text{(EQ 6)}$$

However, as we try to fulfil Equation 3 the converter cannot always serve all of the messages in the buffer queue. Messages cannot be queued too long as Δt_{user} includes both the time in the LB/AC and $\Delta t_{measured}$. To decide whether a message should be accepted to the converter or rejected the following comparison is performed

$$\frac{\sum_{i=1}^{j} N(i)}{p_k} < \tau - th_{k+1} \quad \text{(EQ 7)}$$

to decide which is the largest *j* when the equation is fulfilled. If the first message in the queue is of lower priority than *j* the message is rejected and the comparison is repeated until the first message is accepted.

8. Simulation Parameters

In the simulations 8 SPs and 3 PX-P converters have been used and the value of τ was set to 140 ms. New MakeACall messages were generated according to a four state MMPP with the means 0, 50, 100, 150 calls per second. Changes between the different states occurred according to a poisson process with mean 4 seconds. Two scenarios of arrivals of GetCallInfo messages were investigated, one heavy loaded and one light loaded case. In the heavy loaded case an assumption was made that during a session, the time intervals between generated GetCallInfo messages were exponentially distributed with mean 30 and in the light loaded case the mean was set to 130. The time between the MakeACall and the EndCall message of a session were exponentially distributed with mean 300 seconds. The duration before a call was ended by the network was exponentially distributed by mean 144 seconds.

All PX-P converts were assigned the same capacity of serving 400 SOAP messages per second, which means that $\bar{t}_i = 2,5$ ms. Consequently the service time for a CORBA message was 2.5/4 ms in the converters which means that $C_i = 400$ in the unwrapped case but $C_i \approx 300$ in the wrapped case as a consequence of the processing of the CORBA messages. We set $\bar{x}_{wrapped} = \bar{x}_{unwrapped} = 2,5$ ms, which means that the capacity of the

LB/AC has been decreased by approximately 75% in the wrapped case. The processing time between arrival of a message to the LB/AC until it was queued was set to 1.5 ms. The processing time to send a message from the buffer queue was set to 1 ms independent if it was rejected or accepted.

Three threshold values were used in the LB/AC to decide load level. These were 0.02, 0.06 and 0.08. The maximal value that should not be exceeded, th_{max}, was set to 100 ms. The paces that the load levels resulted in were set to 150, 65, 40 and 30 messages per second in the unwrapped case and 115, 50, 40 and 30 when the wrapped case was considered.

The bucket size and pace r_i were set to 200. The constraints of accepted MakeACall messages per second, d_k was set to 20 for all SPs.

9. Results and Discussion

Of fairness aspects it is important that all LB/ACs accept about the same amount of messages, which means that the LB/ACs must have the same predictions of the load. Our simulations have shown that the different LB/ACs have a rather equal estimate of the current load in the converters. Figure 8 shows a typical comparison of four LB/ACs estimate of the load in a PX-P converter during 50 seconds. It is seen in Figure 8 that when the LB/ACs disagree the estimate about load level is often almost equal. The load levels in the realization shown in the Figure 8 equals in about 75% of the measured points.

Table 1 concludes the outcome from typical simulations of some interesting scenarios. Since the capacity C_i is lower in the wrapped case the load is higher since the same arrival process is used during the simulations independent of whether or not messages are wrapped. It is shown that during heavy load the predicted load levels are equal less often. The high utilization in the scenarios of heavy load that the overload mechanism did not reject more messages than necessary. The reason for the low utilization in the light load scenarios was that there were not messages enough to fully utilize the PX-P converters.

In spite of the overload control mechanism, some of the completed messages were expired. This was mainly caused by the MakeACall messages that were guaranteed. When there was a change from low to high rate of arriving MakeACall messages, the bucket in stage 2 was full. Therefore d_k messages had to be accepted during a very short time

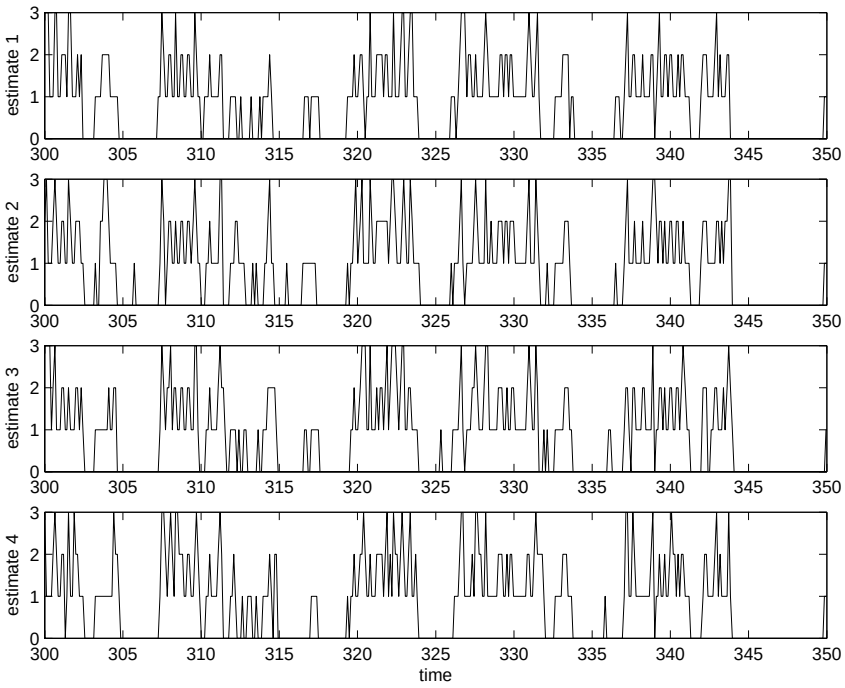


Figure 8. Comparison of four LB/ACs opinion of the predicted load level in a PX-P converter

interval and overload occurred when the converters already were exposed to a high load.

Dependent of how the values of the guarantees d_k is set, we propose that either the contracts should use a shorter time basis or the contracts should be reconsidered to be able to have a complete protection against expired messages.

There are no statistics about the frequencies of the different messages, but predictions by people conversant with the area say that there will be several more GetCallInfos than MakeACall messages. If this is the case and the amount of GetCallInfo messages gets too large, then only the guaranteed amount of MakeACall messages will be accepted as the GetCallInfo messages with higher priority will pad out all other available capacity. Notice in Table 1 how the number of accepted MakeACall messages were decreasing as the relative utilization of the PX-P converters increased.

Table 1. Comparison of the outcome of four different scenarios

	Expired messa-ges (%)	**Utilization of PX-P converters (%)**	**Predicted load equals between different SPs (%)**	**Mean accepted MakeACall messages per SP during 100 s**
SOAP unwrapped heavy load	1.5	99	83	3490
SOAP wrapped heavy load	0.8	100	50	2730
SOAP unwrapped light load	0.8	84	86	4400
SOAP wrapped light load	0.4	88	75	3690

A rate of expired messages around 1% is quite low when the converters are almost fully utilized. This implies that the proposed algorithms where messages are queued in the LB/ACs part instead of queued in the converters is quite effective when considering the aspect of utilization. The common solution of overload control is to make a decision upon the arrival of a message, but we have succesfully proposed an algorithm which tries to delay the decision of admission as long as possible.

The simulations also showed that it was feasible to achieve a functional overload control mechanism without unwrapping the messages to SOAP level in the LB/AC. The constraints for hardware in the LB/ACs could be greatly decreased but instead the constraints for hardware in the PX-P converters increased to be able to achieve the same performance as in the unwrapped case.

10. Conclusions

In this paper we have described a Parlay X application server and its environment and also proposed and evaluated an overload control algorithm for the application server. The overload control mechanism could handle constraints of guaranteed amount of application calls, messages of different priorities and constraints of maximal delay from

request to response. Our results show that it is feasible to achieve overload control at less constraints for hardware if the SOAP messages remains wrapped in the Load Balancing / Admission Control part. The disadvantage is that instead the constraints of hardware in the PX-P converters increase. The rate of expired messages was low, despite the high utilization of the PX-P converters.

The performance seems to be much dependent of the formulations of the contracts. For example will the time base used in the contracts to express the constraints have great impact of the number of served messages with too long delay.

REFERENCES

[1] I. Faynberg; L.R. Gabuzda; M.P. Kaplan and N.J. Shah, “The Intelligent Network Standards: Their Application to Services”, 1st edition, McGraw-Hill, 1996

[2] TINA Consortium, http://www.tinac.com

[3] Parlay Group, http://www,parlay.org

[4] JAIN, http://java.sun.com/products/jain/index.html

[5] 3GPP, http://www.3gpp.org

[6] ETSI, http://www.etsi.org

[7] M Kihl, N Widell, C Nyberg, "Load Balancing Strategies for TINA Networks", In Proceedings of 16th International Teletraffic Congress, Edinburgh, Scotland, June 1999

[8] A W Berger, "Overload control using rate control throttle: selecting token bank capacity for robustness to arrival rates", IEEE Transactions on Automatic Control, vol. 36, 1991

[9] Pham X H, Betts R, "Congestion Control for Intelligent Networks", In proceedings of 1992 international Zurich Seminar On Digital Communications, 1992

[10] M Dahlin, "Interpreting Stale Load Information", IEEE Transactions on parallel and distributed systems, vol. 11, no 10, 2000

[11] Bakshi Y, Diaz A H, Meier-Hellstern K, Milito R A, Skoog R, "Overload control in a distributed system", ITC 15, 1997

[12] Stallings W, Data and Computer Communications, Prentice-Hall, NJ, 2000

[13] ETSI standard 202 915-1 V1.2.1, "Open Service Access (OSA): API; Part 1: Overview", 2003

[14] White paper, "Parlay APIs 4.0; Parlay X Web Services", http://www.parlay.com, Dec., 2002

[15] A Moerdijk, L Klostermann, "Opening the networks with PARLAY/OSA APIs: standards and aspects behind the APIs", IEEE Network Magazine, Vol. 17 Nbr. 3, May, 2003

[16] Chen X, Mohapatra P and Chen H, “An Admission Control Scheme for Predictable Server Response Time for Web Accesses”, In proceeding of 10th WWW Conference, Hong Kong, 2001

Reports on Communication Systems

101 **On Overload Control of SPC-systems**
Ulf Körner, Bengt Wallström, and Christian Nyberg, 1989.
CODEN: LUTEDX/TETS- -7133- -SE+80P

102 **Two Short Papers on Overload Control of Switching Nodes**
Christian Nyberg, Ulf Körner, and Bengt Wallström, 1990.
ISRN LUTEDX/TETS- -1010- -SE+32P

103 **Priorities in Circuit Switched Networks**
Åke Arvidsson, *Ph.D. thesis*, 1990.
ISRN LUTEDX/TETS- -1011- -SE+282P

104 **Estimations of Software Fault Content for Telecommunication Systems**
Bo Lennselius, *Lic. thesis*, 1990.
ISRN LUTEDX/TETS- -1012- -SE+76P

105 **Reusability of Software in Telecommunication Systems**
Anders Sixtensson, *Lic. thesis*, 1990.
ISRN LUTEDX/TETS- -1013- -SE+90P

106 **Software Reliability and Performance Modelling for Telecommunication Systems**
Claes Wohlin, *Ph.D. thesis*, 1991.
ISRN LUTEDX/TETS- -1014- -SE+288P

107 **Service Protection and Overflow in Circuit Switched Networks**
Lars Reneby, *Ph.D. thesis*, 1991.
ISRN LUTEDX/TETS- -1015- -SE+200P

108 **Queueing Models of the Window Flow Control Mechanism**
Lars Falk, *Lic. thesis*, 1991.
ISRN LUTEDX/TETS- -1016- -SE+78P

109 **On Efficiency and Optimality in Overload Control of SPC Systems**
Tobias Rydén, *Lic. thesis*, 1991.
ISRN LUTEDX/TETS- -1017- -SE+48P

110 **Enhancements of Communication Resources**
Johan M. Karlsson, *Ph.D. thesis*, 1992.
ISRN LUTEDX/TETS- -1018- -SE+132P

111 **On Overload Control in Telecommunication Systems**
Christian Nyberg, *Ph.D. thesis*, 1992.
ISRN LUTEDX/TETS- -1019- -SE+140P

112 **Black Box Specification Language for Software Systems**
Henrik Cosmo, *Lic. thesis*, 1994.
ISRN LUTEDX/TETS- -1020- -SE+104P

113 **Queueing Models of Window Flow Control and DQDB Analysis**
Lars Falk, *Ph.D. thesis*, 1995.
ISRN LUTEDX/TETS- -1021- -SE+145P

114 **End to End Transport Protocols over ATM**
Thomas Holmström, *Lic. thesis*, 1995.
ISRN LUTEDX/TETS- -1022- -SE+76P

115 **An Efficient Analysis of Service Interactions in Telecommunications**
Kristoffer Kimbler, *Lic. thesis*, 1995.
ISRN LUTEDX/TETS- -1023- -SE+90P

116 **Usage Specifications for Certification of Software Reliability**
Per Runeson, *Lic. thesis*, May 1996.
ISRN LUTEDX/TETS- -1024- -SE+136P

117 **Achieving an Early Software Reliability Estimate**
Anders Wesslén, *Lic. thesis*, May 1996.
ISRN LUTEDX/TETS- -1025- -SE+142P

118 **On Overload Control in Intelligent Networks**
Maria Kihl, *Lic. thesis*, June 1996.
ISRN LUTEDX/TETS- -1026- -SE+80P

119 **Overload Control in Distributed-Memory Systems**
Ulf Ahlfors, *Lic. thesis*, June 1996.
ISRN LUTEDX/TETS- -1027- -SE+120P

120 **Hierarchical Use Case Modelling for Requirements Engineering**
Björn Regnell, *Lic. thesis*, September 1996.
ISRN LUTEDX/TETS- -1028- -SE+178P

121 **Performance Analysis and Optimization via Simulation**
Anders Svensson, *Ph.D. thesis*, September 1996.
ISRN LUTEDX/TETS- -1029- -SE+96P

122 **On Network Oriented Overload Control in Intelligent Networks**
Lars Angelin, *Lic. thesis*, October 1996.
ISRN LUTEDX/TETS- -1030- -SE+130P

123 **Network Oriented Load Control in Intelligent Networks Based on Optimal Decisions**
Stefan Pettersson, *Lic. thesis*, October 1996.
ISRN LUTEDX/TETS- -1031- -SE+128P

124 **Impact Analysis in Software Process Improvement**
Martin Höst, *Lic. thesis*, December 1996.
ISRN LUTEDX/TETS- -1032- -SE+140P

125 **Towards Local Certifiability in Software Design**
Peter Molin, *Lic. thesis*, February 1997.
ISRN LUTEDX/TETS- -1033- -SE+132P

126 **Models for Estimation of Software Faults and Failures in Inspection and Test**
Per Runeson, *Ph.D. thesis*, January 1998.
ISRN LUTEDX/TETS- -1034- -SE+222P

127 **Reactive Congestion Control in ATM Networks**
Per Johansson, *Lic. thesis*, January 1998.
ISRN LUTEDX/TETS- -1035- -SE+138P

128 **Switch Performance and Mobility Aspects in ATM Networks**
Daniel Søbirk, *Lic. thesis,* June 1998.
ISRN LUTEDX/TETS- -1036- -SE+91P

129 **VPC Management in ATM Networks**
Sven-Olof Larsson, *Lic. thesis,* June 1998.
ISRN LUTEDX/TETS- -1037- -SE+65P

130 **On TCP/IP Traffic Modeling**
Pär Karlsson, *Lic. thesis,* February 1999.
ISRN LUTEDX/TETS- -1038- -SE+94P

131 **Overload Control Strategies for Distributed Communication Networks**
Maria Kihl, *Ph.D. thesis,* March 1999.
ISRN LUTEDX/TETS- -1039- -SE+158P

132 **Requirements Engineering with Use Cases – a Basis for Software Development**
Björn Regnell, *Ph.D. thesis,* April 1999.
ISRN LUTEDX/TETS- -1040- -SE+225P

133 **Utilisation of Historical Data for Controlling and Improving Software Development**
Magnus C. Ohlsson, *Lic. thesis,* May 1999.
ISRN LUTEDX/TETS- -1041- -SE+146P

134 **Early Evaluation of Software Process Change Proposals**
Martin Höst, *Ph.D. thesis,* June 1999.
ISRN LUTEDX/TETS- -1042- -SE+193P

135 **Improving Software Quality through Understanding and Early Estimations**
Anders Wesslén, *Ph.D. thesis,* June 1999.
ISRN LUTEDX/TETS- -1043- -SE+242P

136 **Performance Analysis of Bluetooth**
Niklas Johansson, *Lic. thesis,* March 2000.
ISRN LUTEDX/TETS- -1044- -SE+76P

137 **Controlling Software Quality through Inspections and Fault Content Estimations**
Thomas Thelin, *Lic. thesis,* May 2000
ISRN LUTEDX/TETS- -1045- -SE+146P

138 **On Fault Content Estimations Applied to Software Inspections and Testing**
Håkan Petersson, *Lic. thesis,* May 2000.
ISRN LUTEDX/TETS- -1046- -SE+144P

139 **Modeling and Evaluation of Internet Applications**
Ajit K. Jena, *Lic. thesis,* June 2000.
ISRN LUTEDX/TETS- -1047- -SE+121P

140 **Dynamic traffic Control in Multiservice Networks – Applications of Decision Models**
Ulf Ahlfors, *Ph.D. thesis,* October 2000.
ISRN LUTEDX/TETS- -1048- -SE+183P

141 **ATM Networks Performance – Charging and Wireless Protocols**
Torgny Holmberg, *Lic. thesis,* October 2000.
ISRN LUTEDX/TETS- -1049- -SE+104P

142 **Improving Product Quality through Effective Validation Methods**
Tomas Berling, *Lic. thesis,* December 2000.
ISRN LUTEDX/TETS- -1050- -SE+136P

143 **Controlling Fault-Prone Components for Software Evolution**
Magnus C. Ohlsson, *Ph.D. thesis,* June 2001.
ISRN LUTEDX/TETS- -1051- -SE+218P

144 **Performance of Distributed Information Systems**
Niklas Widell, *Lic. thesis,* February 2002.
ISRN LUTEDX/TETS- -1052- -SE+78P

145 **Quality Improvement in Software Platform Development**
Enrico Johansson, *Lic. thesis,* April 2002.
ISRN LUTEDX/TETS- -1053- -SE+112P

146 **Elicitation and Management of User Requirements in Market-Driven Software Development**
Johan Natt och Dag, *Lic. thesis,* June 2002.
ISRN LUTEDX/TETS- -1054- -SE+158P

147 **Supporting Software Inspections Through Fault Content Estimation and Effectiveness Analysis**
Håkan Petersson, *Ph.D. thesis,* September 2002
ISRN LUTEDX/TETS- -1055- -SE+237P

148 **Empirical Evaluations of Usage-Based Reading and Fault Content Estimation for Software Inspections**
Thomas Thelin, *Ph. D. thesis,* September 2002.
ISRN LUTEDX/TETS- -1056- -SE+210P

149 **Software Information Management in Requirements and Test Documentation**
Thomas Olsson, *Lic. thesis,* October 2002.
ISRN LUTEDX/TETS- -1057- -SE+122P

150 **Increasing Involvement and Acceptance in Software Process Improvement**
Daniel Karlström, *Lic. thesis,* November 2002.
ISRN LUTEDX/TETS- -1058- -SE+125P

151 **Changes to Processes and Architectures; Suggested, Implemented and Analyzed from a Project viewpoint**
Josef Nedstam, *Lic. thesis,* November 2002.
ISRN LUTEDX/TETS- -1059- -SE+124P

152 **Resource Management in Cellular Networks -Handover Prioritization and Load Balancing Procedures**
Roland Zander, *Lic. thesis,* March 2003.
ISRN LUTEDX/TETS- -1060- -SE+120P

153 **On Optimisation of Fair and Robust Backbone Networks**
Pål Nilsson, *Lic. thesis,* October 2003
ISRN LUTEDX/TETS- -1061- -SE+116P

154 **Exploring the Software Verification and Validation Process with Focus on Efficient Fault Detection**
Carina Andersson, *Lic. thesis,* November 2003.
ISRN LUTEDX/TETS- -1062- -SE+134P

155 **Improving Requirements Selection Quality in Market-Driven Software Development**
Lena Karlsson, *Lic. thesis,* November 2003.
ISRN LUTEDX/TETS- -1063- -SE+132P

156 **Fair Scheduling and Resource Allocation in Packet Based Radio Access Networks**
Torgny Holmberg, *Ph.D. thesis,* November 2003.
ISRN LUTEDX/TETS- -1064- -SE+187P

157 **Increasing Product Quality by Verification and Validation Improvements in an Industrial Setting**
Thomas Berling, *Ph.D. thesis,* December 2003.
ISRN LUTEDX/TETS- -1065- -SE+208P

158 **Some Topics in Web Performance Analysis**
Jianhua Cao, *Lic. thesis,* June 2004.
ISRN LUTEDX/TETS- -1066- -SE+99P

159 **Overload Control and Performance Evaluation in a Parlay/OSA Environment**
Jens K Andersson, *Lic. thesis,* August 2004.
ISRN LUTEDX/TETS- -1067- -SE+100P