Citation for the published paper:
J. Andersson and M. Kihl, 2004,
"Load Balancing and Admission Control of
a Parlay X Application Server",
*Seventeenth Nordic Teletraffic Seminar, NTS 17, Fornebu, Norway,
25-27 August 2004*.
ISBN: 82-423-0595-1. Publisher: Fornebu : Telenor.

LOAD BALANCING AND ADMISSION CONTROL OF A PARLAY X APPLICATION SERVER

JENS ANDERSSON AND MARIA KIHL
LUND UNIVERSITY, SWEDEN
DEPARTMENT OF COMMUNICATION SYSTEMS
email: {jens.andersson, maria.kihl}@telecom.lth.se

Abstract

To increase the number of services and applications in the telecommunication networks a new service architecture has been proposed and specified. The consortia that initiated the new service architecture is called Parlay. Since then, there has been a continued development and 3GPP the standardization group for 3G networks, is now cooperating with Parlay. 3GPP and Parlay are together specifying the Parlay/OSA standard.

However, the aim with the new service architecture is to make the art of service development so easy that any software developer should be a potential application developer for telecommunication networks. It is foreseen to be a dramatic increase of new services, but it has been shown that it is still complex to create new applications. Parlay X is a standard which is an extension of Parlay/OSA, initiated to allow access to the telecommunication network capabilities via web services. The Parlay X architectures include an Application Server (AS), which translates the web service calls to Parlay/OSA commands. A typical AS is a distributed environment sensitive to overload.

In this paper we propose and investigate an overload control mechanisms for an AS. The methods have been implemented in a real AS to evaluate the performance. Parlay X is a contract driven architecture, where different service providers have different constraints about minimum number of service calls served per second and maximal delays of different services. The methods and algorithms are designed to serve different service providers with different contract parameters.

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.

Parlay [2] and JAIN [3] 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 [4]. The Parlay/OSA APIs are standardized, both by ETSI [5] 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 capability by a single command via a so called Application Server (AS). The AS is a distributed system that performs the mapping between Parlay X commands and Parlay/OSA commands.

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 load balancing and admission control methods are well known concepts for congestion avoidance. Berger [6] has a discussion about different admission control methods and Pham et al. [7] gives an example of an overload control mechanism for IN. Dahlin [8] discusses what impact old information about the load status should have on the decisions.

This paper gives an overall description of the Parlay/OSA and Parlay X service architecture. It is focused to describe the Application Server and its functionality when considering the Parlay X standard. We will also present an overload control algorithm that is designed for a distributed Application Server with messages of different priorities. The performance of the algorithms and methods is investigated by real measurements.

2. 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 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 [10].

2.1 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 paper. 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 [11].

The architecture of a typical Parlay/OSA environment connected to the Internet can be seen in Figure 1. 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 in a Parlay X environment 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.

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

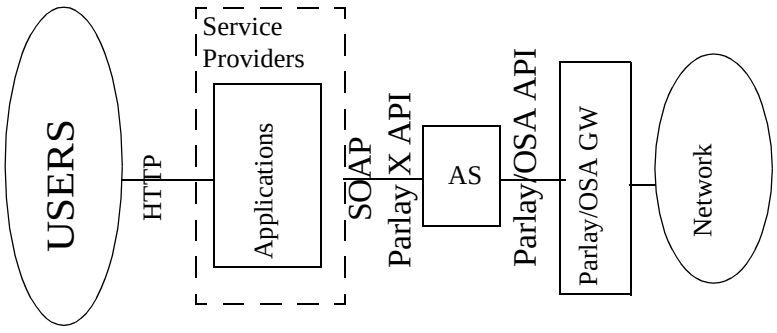


Figure 1. Architectural picture of an AS and Parlay/OSA environment

Parlay/OSA standards, but it depends on the underlying network what SCFs that are supported. More about the Parlay/OSA gateway can be found in [12].

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. The ease of use and the lucidity of the applications will increase when they are presented on a monitor and controlled by text input.

2.2 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. Constraints are often divided into hard or soft constraints. Hard constraints should always be fulfilled, and soft constraint is more like a guideline. An example of a hard constraint is the defined amount of application calls from a certain SP that at least should be accepted each time unit. The time constraint for the maximal delay is a soft constraint. 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.

3. THE APPLICATION SERVER

A detailed view of an AS is shown in Figure 2. 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 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 performed 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

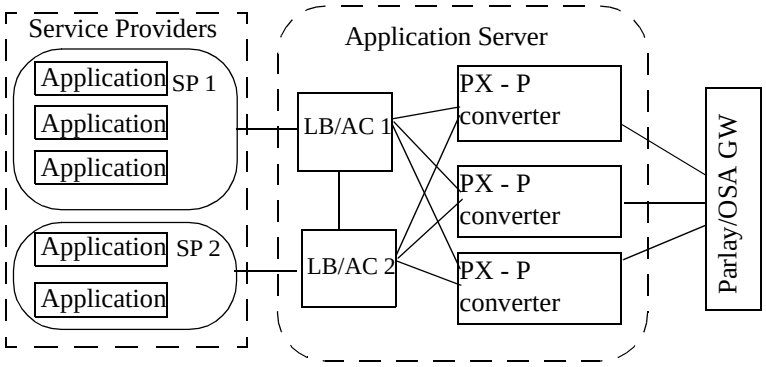


Figure 2. Detailed view of the AS

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. The calls can either be ended by an EndCall message or from the network side (e.g. the parties hang up), without noticing the AS.

3.1 Load Control Mechanism

The load control mechanism can be described by Figure 3. 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 section 3.3. 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 are 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.

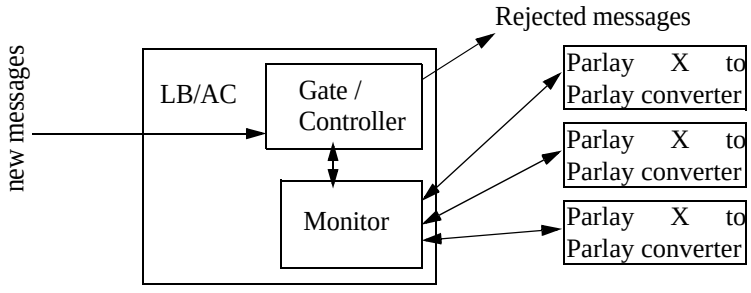
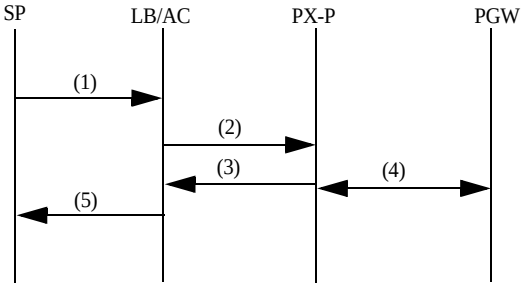


Figure 3. The load control mechanism



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

Figure 4. Sequence diagram of MakeCall

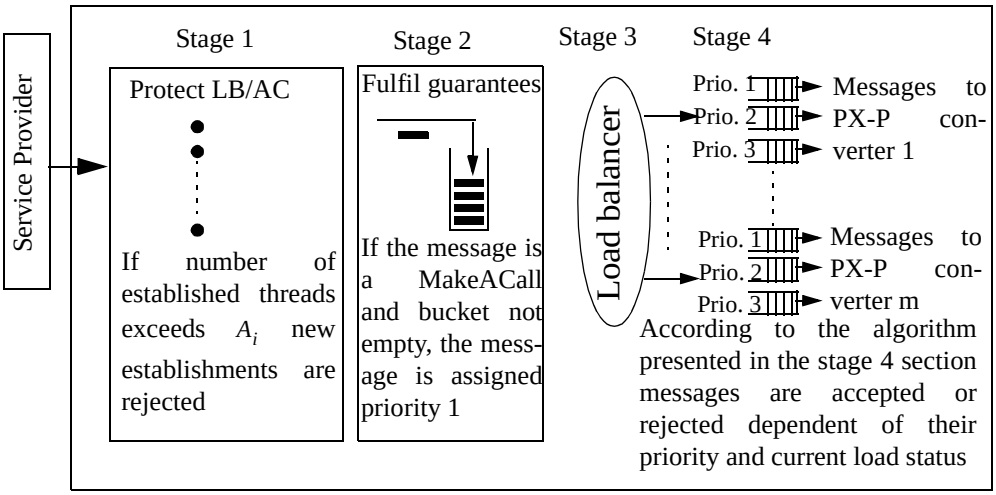
3.2 Objectives of this paper

The objectives are to investigate how an overload control algorithm can be implemented to maximize the throughput and to get a robust system at the same time as the constraints are considered. Each SP should have a guaranteed amount of accepted MakeACall messages each time unit. There are possibilities that the PX-P converters will contain a database or other software that require processor capacity and thereby may cause transients of load when the converter is occupied by other processes. The algorithm is implementation in a real environment and real measurements of the proposed mechanism should be compared with a traditional overload control mechanism.

3.3 Example of a MakeACall Request

In Figure 4 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 (4) involves different amount of communication dependent of how many participants the SOAP message attend to connect by the MakeACall message. The messages sent between the PX-P converter and the PGW are CORBA messages. Typically it takes 5 CORBA messages to connect two parties. The communication with the PGW consists of calling appropriate Service Capability Features (SCFs).

The sequence diagram for a GetCallInfo message is almost the same as shown in Figure 4 excluding message (4). An EndCall message has a sequence diagram identical to Figure 4, where (4) correspond to 1 CORBA message.



igure 5. Overview of the different stages in the LB/AC

4. OVERLOAD CONTROL

In this section we will propose an overload control mechanism for robust protection of an AS. 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 4. 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 5.

4.1 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. LB/AC *i* only agrees to have A_i simultaneous threads to the SP. If the SP tries to establish an additional thread the establishment is rejected without noticing the SP. In the contracts a maximal number of messages sent per second, denoted r_i will be agreed for the SP. If this number is exceeded, correct treatment of priorities and notification of rejecting cannot be guaranteed. As the time constraint for a message was set to τ we can assume that this is the maximal time a thread is connected between the LB/AC and the SP. If the LB/AC shall treat r_i messages per time unit the number of threads needed can be calculated as follows

$$A_i = \tau \cdot r_i \qquad \text{(EQ 1)}$$

Stage 1 simply consist of an active counter of number of established threads.

4.2 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_i accepted MakeACall messages per second is fulfilled. To fulfil the guaranteed number of accepted MakeACall messages for SP *i* we use a token bucket of size d_i and tokens arriving with rate d_i. 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 thereby it 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. The GetCallInfo and EndCall messages are always forwarded with their original priorities 2 and 1.

4.3 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. 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. 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. 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.

4.4 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

$$\Delta t_{user} < \tau \qquad \text{(EQ 2)}$$

Proposed admission control algorithm

The following algorithm is designed to be sensitive to transients in the load condition of the PX-P converters. To avoid buffering messages at the PX-P converters during overload it is

only allowed to have one active thread between an LB/AC and a PX-P converter. When a message is load balanced it is sent to a priority queue. In the considered case with messages of three priorities, each LB/AC have three queues for each PX-P converter, see Figure 5. Statistics are maintained on how many messages of priority *j* that are present in a certain queue and this number is denoted $N(j)$. The messages in the queues are assigned a time stamp when the message are put into the queue. By using the time stamp information can be maintained of how long a message has been queuing in the LB/AC, denoted Δt_{queued}. To predict the load in the converters the measured times $\Delta t_{measured}$ are used. $\Delta t_{measured}$ is actually expressing the load status when the message was sent to the converter, but we assume that the load will not change too fast.

However, as we try to fulfil Equation 2 the converter cannot always serve all of the messages in the queues. Messages cannot be queued too long as Δt_{user} includes both Δt_{queued} and $\Delta t_{measured}$. So first of all the messages, excluding priority 1 messages, that do not fulfil the following constraint are rejected

$$\Delta t_{measured} + \Delta t_{queued} < \tau \cdot f \qquad \text{(EQ 3)}$$

as the service time for a message is assumed to be $\Delta t_{measured}$. The variable *f* is just a factor less than 1, used to have a margin. Then the following comparison is performed to predict from how many priority queues we can accept messages if the total time in the PX-P converter, $\Delta t_{measured}$, remain the same and the constraint should be fulfilled

$$\sum_{j=0}^{m} N(j) \cdot \Delta t_{measured} < \tau \cdot f \qquad \text{(EQ 4)}$$

The largest *k* for which the condition is fulfilled equals the highest priority that can be accepted. Now the message that has been queuing longest in the LB/AC is chosen from any of the queues with messages with a $\text{priority} \leq j$.

5. Configuration and setup parameters

In the simulations 4 SPs, 4 LB/ACs and 2 PX-P converters have been used. The equipment used during the investigations was provided by Appium AB. The processing time for handling a SOAP message (unwrapping and wrapping) in a PX-P converter is measured to be about 2,5 ms. The processing time for handling a CORBA message is measured to be about 2,5 / 4 ms. This mean that the total service time in the converter for an EndCall message is 12,5 / 4 ms.

There are reasons to believe that the traffic to ASs will be rather bursty, since the AS in Parlay X are 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. [13]. We believe that MMPP is a good assumption also in the context of ASs, and this scenario will be used to compare the proposed algorithm with other algorithms. New MakeACall messages were generated according to a four state MMPP with the means 25, 33, 50, 67 calls per second. Changes between the different states occurred according to a poisson process with exponential distributed time

intervals with mean 4 seconds. GetCallInfo messages are generated to a certain call-ID with exponential distributed time intervals with mean three times higher than the MakeACall messages. As no calls will be ended from the network in our setup EndCall messages are generated with the same intensity as MakeACall messages. The behaviour of the proposed overload control mechanism is also evaluated during steady state poisson arrivals with means 10, 25, 33, 50 and 67 calls/s.

The constraints of accepted MakeACall messages per second, d_i was set to 25 for all SPs. The value of τ was set to 250 ms and *f* was set to 0,9. A_i is set to 25 for all of the LB/ACs.

6. Results and discussion

The proposed algorithm has successfully been implemented in a real Parlay X environment. Comparisons of the proposed overload control mechanism in this paper is made with the algorithm proposed in Andersson et. al. [14], and also with a common used algorithm in the context of a system with priorities and time constraints.

6.1 Results during different load scenarios

The result during the different arrival processes described in section 5 is concluded by Table 1. From the results it is seen that the mechanism react and rejects more messages as the load increases. As mentioned before it is not only by processing the messages that the PX-P converters get loaded. Figure 6 shows how the service times varies over the time. The transients in the figure is not a consequence of too many messages in the converters. Only 4 SPs are used, which result in a maximum of 4 messages at the same time in a converter. The sudden peaks are created when a converter starts to process other processes. This is also the reason why some of the messages are expired. As none of the priority 1 messages are rejected the priority 1 queue gets long when a converter is occupied with other processes. Assume that

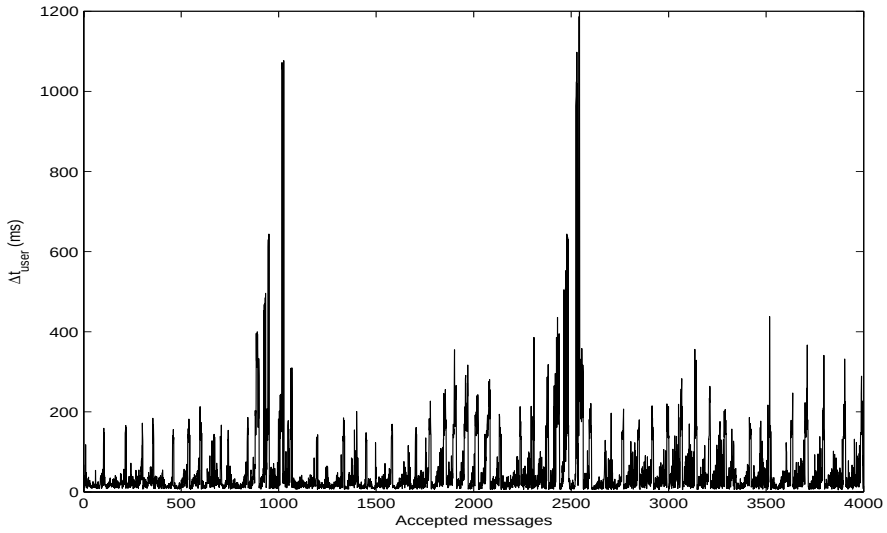


Figure 6. Typical outcome of Δt_{user} during a realisation.

Table 1: Outcome during different scenarios of arrivals

100 s simulation scenarios	Steady state 10/s	Steady state 25/s	Steady state 33/s	Steady state 50/s	Steady state 67/s	MMPP proposed mechanism
rejected prio. 2 messages	0,1%	0,5%	0,9%	3%	11%	4%
Rejected prio. 3 messages	0%	0%	4%	15%	35%	17%
tot. accepted MakeACalls	1000	2400	3100	3400	3500	3300
tot. accepted GetCallnfos	3000	6900	8400	9100	7000	7600
tot. accepted EndCalls	1000	2400	3100	3300	3300	3200
Mean service time	10ms	19ms	19ms	70ms	127ms	65ms
Expired messages	0.2%	1%	1%	6%	14%	5%

a converter is busy for over 250 ms. Then the priority 1 queue may get full during a heavy load scenario. Therefore 25 messages are expired each time something like that happen. Despite that, the rate of expired messages is quite low. From the table it is also seen that when enough messages are provided, there is always more than 2500 MakeACall messages accepted (the guaranteed value during 100 seconds). This indicate that stage 2 in the algorithm function sufficient.

6.2 Comparison with other algorithms

The algorithm proposed in Andersson et. al. [14] is quite similar to the one in this paper. The algorithm presented in this paper can yet deal with a system where other processes are prioritized and a converter may be unavailable during a time interval. As the proposed algorithm only use one thread it will have instant feedback to stop sending messages. With the mechanism presented in [14] such a scenario only result in a lower rate of messages. Too many messages queued at a converter seem to result in an unpredictable behaviour, not preferable. Measurements have shown bad results of expired messages with that algorithm.

Measurements of a traditional overload control mechanism has also been performed for comparison. This traditional mechanism make the decision whether a message should be rejected or accepted at once when a message arrives. This mechanism cannot guarantee the amount of MakeACalls but it does consider priorities. The decision of rejection is based on how many messages that have been sent during the preceding time interval. A similar stage 1 rejection mechanism as in the proposed mechanism is also included in this mechanism. When measurements are performed with the traditional mechanism during the MMPP arrival process the following results are obtained: 5% expired messages, a meantime of 68 ms, 3000

MakeACalls accepted, 7000 GetCallInfos accepted and 2850 EndCalls accepted. These values compared to the results for the proposed mechanism show how we gain in throughput, compare with Table 1. The throughput increases as messages are buffered waiting to see if the current load is decreasing such that it is feasible to succeed the deadline despite heavy load at the moment.

7. 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. When comparing the proposed mechanism with a classical mechanism the rate of expired messages and mean time for service was about same, but the total number of accepted messages during 100 seconds clearly differed to advantage for the proposed method. This is the gain by buffering messages in the proposed method, compared to the classical method where decisions are taken immediately at an arrival.

Acknowledgement

The authors would like to thank Christian Zieger and Per Karlsson for their help with implementation and measurements. This work was financially supported by VINNOVA proj. nbr. 23918-2.

REFERENCES

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

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

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

[4] 3GPP, http://www.3gpp.org

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

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

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

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

[9] Stallings W, 2000. *Data and Computer Communications*, Prentice-Hall, NJ

[10] ETSI standard 202 915-1 V1.2.1, 2003. " Open Service Access (OSA): API; Part 1: Overview", http://www.3gpp.org

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

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

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

[14] Andersson J, Kihl M and Söbirk D, 2004. “Overload Control of a Parlay X Application Server”, In proceeding of SPECTS 2004, San Jose