Citation for the published paper:
J. Andersson, M. Kihl, and C. Nyberg, 2003,
"Priorities and overload control in OSA",
Fourth International Conference on 3G Mobile Communication Technologies (3G 2003) : 25 - 27 June 2003, Savoy Place, London, UK (IEE Conference Publications ; vol. 494)
ISBN: 0-85296-756-X.
Published with permission from: IEE (Institution of Electrical Engineers).
http://dx.doi.org/10.1049/cp:20030353

PRIORITIES AND OVERLOAD CONTROL IN OSA

J K Andersson, M Kihl and C Nyberg

Lund Institute of Technology, Sweden

ABSTRACT

In this paper, an OSA gateway is modelled in a multi application environment. Also, an overload control mechanism is proposed. The mechanism supports different time constraints and priorities for the applications. By simulations it is shown how the proposed overload control works during different load conditions in the modelled environment.

INTRODUCTION

During the last years there has been a change in the evolution of service architectures. As it is now, each network has its own service architecture and only the operator is able to create 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 capabilities from other networks. Service creation will also be much easier in the new architectures.

Open Service Access (OSA) is the service architecture that is proposed for the 3G networks. OSA is developed by the 3GPP (7). With OSA it becomes easier to develop and test new services outside the telecom domain. Since OSA offers an increased security and integrity, operators may dare 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. Overload control of communication systems has been a research topic for some decades in telephone networks. An early paper is Forys (8) in which the protection of control processors in telephone exchanges is discussed. Some papers on overload control in IN are Pham and Betts (9) and Kihl and Nyberg (10) in which overload control algorithms are suggested and investigated. Until now, no papers have discussed overload control in OSA. The general performance of a Parlay gateway, which is almost the same as an OSA gateway was analysed in Melen et al (11).

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 operators 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 may be weighted according to, for example, the amount of revenue the application generates for the operator.

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. Also, we develop an overload control mechanism with priorities, in which the main objective is that all admitted application calls should keep a certain deadline.

OPEN SERVICE ACCESS (OSA)

OSA consists of three parts, the Application Servers (AS:s), the Service Capability Servers (SCS:s), and the Framework. Fig. 1 shows one of the possible configurations of an OSA architecture. Each SCS hosts the 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 several physical entities. In Fig. 1 the Framework and both the SCS:s constitute the OSA gateway.

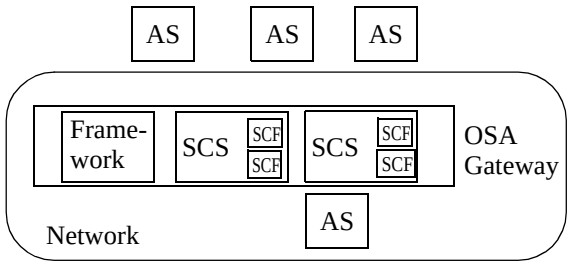

Figure 1. An OSA architecture.

The Application Servers (AS:s) host the applications. The applications can be Virtual Private Networks (VPNs), conferencing, location based applications and so forth. Each AS can host one or several different applications. The AS:s can be physically positioned inside or outside the network they are communicating with.

In an OSA architecture there can be one or several Service Capability Servers (SCS:s). The SCS provides the applications with network functionality via one or several SCF:s. An SCF consists of several narrow functions, which together render 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. For more details about SCS:s, see Stretch (3).

The Framework can be seen as a separate SCS providing the applications with basic mechanisms, like authentication before accessing the network functionalities or discovery to find out which SCF:s that are provided by the SCS:s. Also, the Framework supplies security and integrity functionalities.

An example of a service in OSA

In (4), a general OSA application is developed. It is an "application initiated call", where 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 Fig. 2. An application setup consists of a number of OSA messages. First the application sends a createCall message to the SCS to create objects for further communication. In the SCS 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 now the call is routed to the B party.

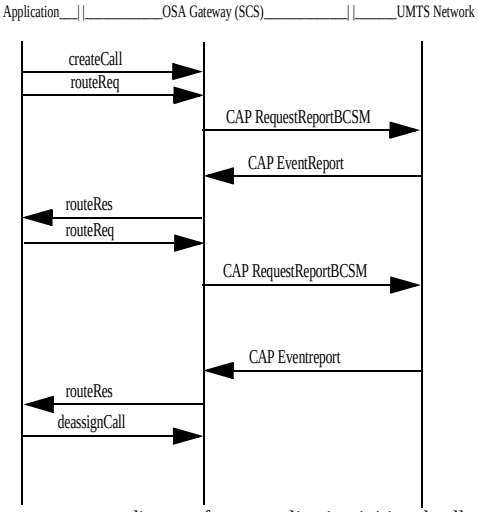

Figure 2. Message sequence diagram for an application initiated call.

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 specifications (5), 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. Intuitive the simplest way is to let the AS take care of its own load control and the Gateway take care of its.

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

Contract writing

When a new application is introduced it signs a contract with the OSA gateway through the Framework. Proposals of what a contract should include can be read for example in (2). A typical contract might include a variable determining how many application calls an application at least should have accepted each second and a variable determining the maximum 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.

MODEL

We have modelled a Gateway consisting of one SCS containing one SCF in a multi application environment. Our model consists of *R* applications, a call control SCF and a network, see figure 3.

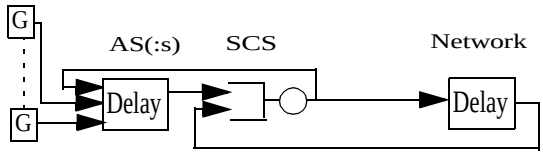

Figure 3. An OSA Model.

New application calls are generated as a Poisson process, each with their own arrival rate. Each G-box in figure 3 corresponds to the generator of new application calls to a specific application. Each application is assumed to be positioned at a non-overloaded AS and is modelled as a delay with deterministic values that vary depending of which message to execute.

An application, a_l, has a guaranteed rate of d_l calls per second, and a total execution time in the SCS of $\bar{x}_{tot}(l)$ seconds. Of course the system must be stable which implies that

$$\sum_{l=1}^{R} d_l \cdot \bar{x}_{tot}(l) < 1 \qquad (1)$$

The 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, where p_j guarantees a higher rate than p_{j+1}. A time constraint T_k corresponds to the maximum delay a message should experience each time it passes the SCS. In the example in figure 2 the application call has to pass the SCS five times and if the application call, when it is completed, has had a total residence time in the SCS 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 independent of each other to each application.

The SCS is modelled as a single server queue, in which one message at a time is served. The execution times in the SCS are set to deterministic values. We have assumed that the deviation in the execution times is small. We denote with x_{scs} the largest execution time in the SCS, and with $N(t)$ the number of messages in the SCS with a remaining time *t* before their deadline expires.

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 Fig. 2. Also, we have assumed that all applications require the same execution as in Fig. 2.

OVERLOAD CONTROL MECHANISMS

We have developed an overload control mechanism for the SCS, shown in figure 4. It consists of a controller, a gate and a selector. The *controller* makes appropriate measurements on the SCS. Also, it analyses the measurement data and determines what action that has to be taken by the *gate*, which regulates the acceptance of new application calls.

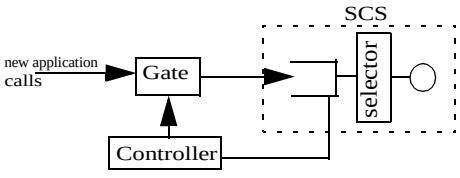

Figure 4. Model of the overload control mechanism

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 SCS should be served. The selector uses an Earliest Deadline First (EDF) scheduling algorithm, see Lui and Layland (1). The controller performs measurements of the load status in the SCS to check if it is possible at all for the selector not to exceed the time constraints. If not, the controller informs 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.

Gate

When there is an overload situation in the SCS, the gate starts to reject application calls. The gate uses a call gapping method, see Berger (6), to reject application calls. The time is divided into small intervals of a certain length, and then exactly one application call is accepted each interval. The interval lengths are dependent of which 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, call gapping is introduced on application calls of the next priority level, and so on until applications from all priority levels have their calls rejected according to the call gapping method. 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.

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:

$$N(T_k) \cdot x_{scs} \le T_k, \quad (1 \le k \le N) \qquad (2)$$

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.

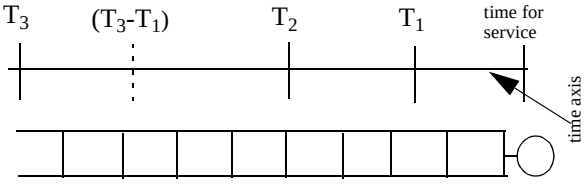

Figure 5. Abstraction of controller mechanism

Figure 5 shows an abstraction of the controller mechanism with three time constraints. Each time constraint can be seen as an insertion point in the waiting queue. 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, condition (2) can be improved to also include the guaranteed rate of new application calls. This condition can be expressed as

$$\left(N(T_k) + \left(\sum_{j=1}^{k-1} \lambda_j \cdot (T_k - T_j) \right) \right) \cdot x_{scs} < T_k \qquad 1 \le k \le N \quad (3)$$

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

However, the controller should also check that the condition in (3) 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 calls from all priority levels are rejected and a burst of application calls with time constraint T_i arrives, then the maximal number of messages that might be additional to an interval of length T_k can be expressed as

$$\sum_{j=k}^{i-1} \lambda_j \cdot T + \sum_{j=1}^{k-1} \lambda_j \cdot T_j \qquad \begin{matrix} 2 \le i \le N \\ i > k \end{matrix}$$
$$T = \begin{cases} T_k & T_k < (T_i - T_j) \\ (T_i - T_j) & T_k \ge (T_i - T_j) \end{cases} \quad (4)$$

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

$$\left(N(T_i) - N(T_i - T_k) + \left(\sum_{j=1}^{k-1} \lambda_j \cdot (T_k - T_j) \right) + \right. \quad (5)$$
$$\left. \sum_{j=k}^{i-1} \lambda_j \cdot T + \sum_{j=1}^{k-1} \lambda_j \cdot T_j \right) \cdot x_{scs} < T_k \qquad \begin{matrix} 2 \le i \le N \\ i > k \end{matrix}$$

where T is the same as in equation (4). This constraint has to be fulfilled for all possible combinations of T_k and T_i, where $i > k$.

If conditions (3) and (5) 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.

SIMULATION PARAMETERS

The SCS is modelled as a single server queue with capacity of serving 100 application calls per second, assumed that each application call requires the same execution as the example in Fig. 2. Each time a message executed in the SCS results in a new message, the service time is 2 ms. Otherwise, the service time is 1 ms. Each delay in the AS is 1 ms.

Each call has to executed twice in the network. The first time is when the call is connected to the callers phone. In this case a delay of 10 ms is used, since there is probably some kind of auto phone pick up function. The second execution correspond to the B party pick up time. This pick up time exponentially distributed with mean 2 seconds.

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. Also, there is a minimum time of 50 ms between changes.

To evaluate a characteristic behaviour of the proposed overload mechanism, we have investigated two different configurations of the application variables. In configuration 1 there are three different applications with three different priorities and two different time constraints. Application a_1, a_2, and a_3 correspond to priority 1, 2, and 3 respectively. The priorities in increasing order correspond to guaranteed rates of 50, 10, and 0.5 application calls per second. a_1 and a_3 have a time constraint of 100 ms, and a_2 has a time constraint of 1 second.

Configuration 2 has 10 applications divided into three priority classes and two time constraints. Priorities 1, 2, and 3 correspond to guaranteed rates of 10, 5, and 0.5 calls per second, respectively. Applications 1, 3, 5, 7, and 9 have a time constraint of 100 ms and the other applications have a time constraint of 1 s. Applications 1, 4, and 5 have priority 1. Applications 2, 6, 7, and 8 have priority 2. Consequently, applications 3, 9, and 10 have priority 3.

RESULTS AND DISCUSSION

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

of an overload mechanism as proposed is discussed.

We first consider configuration 1 and let the arrival rates, λ, for each application increase every 25th second. In Fig. 6 the resulting rates of completed application calls are shown. As long as the total arrival rate is below the capacity of the SCS, no calls are rejected. However, after 50 seconds, the total arrival rate exceeds the capacity of the SCS and, therefore, application calls with the lowest priority are rejected. After 75 seconds, calls from all priority levels are rejected, however all applications will have their guaranteed rate of application calls. In the realization shown in figure 6 about 0.2% of the served application calls were so-called expired calls. The SCS has a utilization of 88% during the 100 seconds. If the simulation should have been done such that the total λ is larger than the capacity of the SCS the utilization is almost 99% instead.

It is interesting to see what we gain with a priority based rejection mechanism as proposed. An estimation of how good the outcome is can be performed if we introduce a utility measure *U*. As an example, we can assume that priority 1, 2, 3 correspond to utility 3, 2, 1 respectively. In the realisation shown in Fig. 6,

$$\int U dt \approx 19400$$

If an ordinary random rejection method would be used, *U* would have an upper bound at 17600.

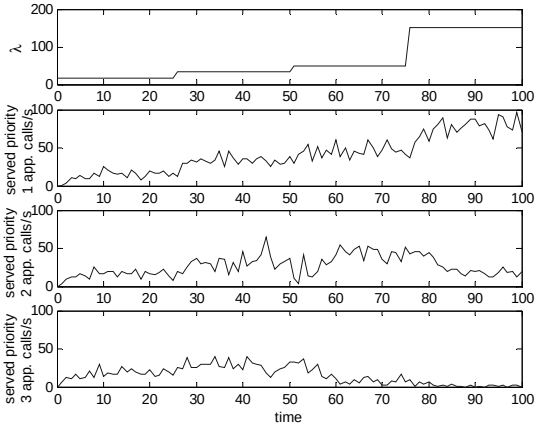

Figure 6. Results for configuration 1.

Next, we use configuration 2 and let the applications randomly adopt a λ that equals 0, 10, 20 or 30 calls per second, which is kept during an exponentially distributed time with mean 2 seconds. In this case, the rate of expired calls will be less than 0.1%. The utilization of the SCS in this case is larger than 97%. The utility measure for a simulation of 100 seconds becomes.

$$\int U dt \approx 21100$$

If all applications would have been treated equally, *U* would have an upper bound at 20000. Clearly we get a better gain when the priority based overload control is used.

CONCLUSION

We have modelled an overload control mechanism for an SCS in an OSA architecture. The overload control mechanism can implement priorities among applications. The overload control mechanism also makes sure that application calls that are accepted will meet their time constraint with a high probability.

Also, we have shown that the total gain of the served application calls is higher when we use our 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. Liu C. L. and Layland J. W., 1973, “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment“, Journal of the Association for Computing Machinery, Vol. 20 No. 1, 46-61

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

3. Stretch R. M., 2001, “The OSA API and other related issues”, B T Technol J., Vol. 19 No 1, 80-87

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

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

6. Berger A., 1991, "Comparsion of Call gapping and Percent blocking for overload control in distributed switching systems and telecommunications networks", IEEE Transactions on Communications, vol. 39, 407-414

7. The 3GPP home page, "www.3gpp.org"

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

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

10. Kihl M. and Nyberg C., 1997, "Investigation of overload control algorithms for SCPs in the intelligent network", Communications IEE Proceedings, vol. 144, 419-423

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