Adaptive Load Control Algorithms for 3rd Generation Mobile Networks

Simin Nadjm-Tehrani[1],Kayvan Najarian[2],Calin Curescu[1],Tomas Lingvall[3],Teresa A. Dahlberg[2]

[1]Dept. of Computer and Information Science, Linköping University, Sweden

[2]Dept. of Computer Science, University of North Carolina at Charlotte, USA

[3]Center for Radio Network Control, Ericsson Radio Systems, Linköping, Sweden

{simin,calcu}@ida.liu.se {knajaria,tdahlber}@uncc.edu

ABSTRACT

In this paper we present strategies to deal with inherent load uncertainties in future generation mobile networks. We address the interplay between user differentiation and resource allocation, and specifically the problem of CPU load control in a radio network controller (RNC).

The algorithms we present distinguish between two types of uncertainty: the resource needs for arriving requests and the variation over time with respect to user service policies. We use feedback mechanisms inspired by automatic control techniques for the first type, and policy-dependent deterministic algorithms for the second type. We test alternative strategies in overload situations. One approach combines feedback control with a pool allocation mechanism, and a second is based on a rejection-ratio-minimising algorithm together with a state estimator.

A simulation environment and traffic models for users with voice, mail, SMS and web browsing sessions were built for the purpose of evaluation of the above strategies. Our load control architectures were tested in comparison with an existing algorithm based on the leaky bucket principle. The studies show a superior behaviour with respect to load control in presence of relative user priorities, and minimal rejection criteria. Moreover, our architecture can be tuned to future developments with respect to user differentiation policy.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *Wireless communication, packet-switching network.* C.2.1 [**Computer-Communication Networks**]: Network Operations – *Network Management.* I.6.4 [**Simulation and Modelling**]: Model Validation and Analysis.

General Terms

Algorithms, Design.

Keywords

Resource allocation, load control, QoS

MSWiM'02, September 28, 2002, Atlanta, Georgia, USA.

1. INTRODUCTION

Providing Quality of Service (QoS) [3] towards network users has been present ever since the first networks for voice or data communication were conceived. What is new in our day and age is the rebirth of several earlier techniques when telecommunication and IP networks meet.

Several authors put forward the scenario that IP will be the common ingredient of future networks, and thereby base the study of QoS issues on the techniques developed for Internet traffic [2]. Others study the special characteristics of mobile communication in presence of IP-based services [10,14]. In this paper we consider how enforcing a simple form of QoS in mobile networks can be integrated into a resource allocation problem, namely the CPU load control algorithms.

Third generation (3G) mobile networks have a number of aspects common with IP networks: the complexity of traffic models for users of services such as mail, short message services (SMS), and web browsing. In addition, they have to deal with a new factor that affects the offered load to the system: the user mobility. These factors together lead to execution of various functions in the radio control node in order to dynamically allocate data and control channel (types and rates), set up and tear down connections, and so forth. Thus, 3G networks need adaptive resource management for different resource types that need to be allocated inter-dependently.

A major track of research is the allocation of bandwidth as a resource and adaptive admission control algorithms based on available bandwidth and QoS requirements [4,7,8,11]. Another resource type is the CPU, which processes the generated tasks resulting from the above dynamic decisions after a user has been admitted to a system. As user needs change, and the available bandwidth is increased/decreased, the results of the bandwidth allocation algorithm affect the load on a CPU that is executing the required control functions. Note that even operating at maximum bandwidth allocation levels there will be new incoming tasks for the CPU. Therefore dealing with the more scarce resource (the bandwidth) will not eliminate CPU overloads. To solve this problem with over-provision will be wasteful since such overloads should not be too common. If the CPU overload is allowed a hardware based resetting mechanism will be activated, with an even worse effect on system availability. Thus we need to control the CPU utilisation during overloads. This paper presents the special algorithms to ensure that a radio control CPU is not overloaded during traffic overload situations.

The result of load control algorithms may be the rejection of some functions. Hence, a differentiation scheme (perhaps similar to the

one used in the bandwidth allocation algorithm) needs to be in place. As bandwidth allocation schemes use more complex user profiles as input, we aim for a consistent solution to the load control problem. System-wide optimal resource allocation needs to take care of interdependencies between these two resource types, and our work is a step towards understanding some of the interdependencies.

A subtle complexity in the application of algorithms for adaptive resource allocation together with user differentiation is the mutual effect of the network algorithms on user behaviours. We therefore propose a framework in which a mix of techniques from real-time systems, control theory and AI can be applied and simulated.

1.1 Related areas of work

The work on adaptive resource allocation in networks spans a number of research communities. In this section we provide an overview of the areas based on selected works from each area.

Some proposals for QoS-based resource allocation in 3G have a fully adaptable scheme, where each connection can be run with different QoS levels, with a goal to maximise the total benefit. Others offer strict guarantees and reservations to fulfil a QoS contract. Lately these two techniques are merging, connections can request a minimum bandwidth, which has to be guaranteed, and a maximum bandwidth, which gives the highest local reward [7]. Richardson et al. [12] propose an algorithm to schedule a downlink in a 3G network. Priority-based weighted fair queuing [15] combines priority-driven and share-driven scheduling which lowers the delay of important sessions while maintaining the guaranteed bandwidth. Another ongoing work addresses developing adaptive resource management protocols to maintain QoS guarantees in the wake of network failures [5,6]. This work focuses on the lowest network layers for managing dynamic use of wireless channels by users with multiple service guarantees. None of these works treat CPU load control.

Over two decades of research in the real-time systems community treats the question of CPU's load in general, and scheduling of jobs with well-defined characteristics in particular. Recently, the area of automatic control in which feedback is a means of dealing with uncertainties, has influenced the work on adaptive scheduling. This provides powerful tools for deriving analytical results in presence of certain assumptions [1,13]. Stankovic et al [13] present a distributed system where feedback control is used for minimizing deadline miss ratio at over-utilisation, and keep CPU utilisation as high as possible at under-utilisation. The system is purely adaptive; no priority between tasks can be specified. Abeni et. al [1] present a feedback adaptive QoS scheme which integrates the notion of constant bandwidth server (CBS) into an earliest-deadline-first (EDF) scheduling environment. The CBS provides the temporal isolation for the different tasks, while adaptivity is performed globally by increasing the CPU share of a task, or locally by constraining the application to a given bandwidth. While there is a notion of priority between the tasks, it does not guarantee a certain CPU share. This work is strongly tied up with the EDF scheduling policy. Note that in our case the load control algorithm decides about the incoming tasks independently from the scheduling policy used by the underlying operating system.

2. LOAD CONTROL IN 3G NETWORKS

The Radio Network Controller is responsible for controlling the common resources in a 3G radio access network and serving the User Equipment (UE) in the best possible way. The hardware in each RNC consists of a number of processors each responsible for handling UEs within a certain part of the network. A user request arrives at one of these CPUs to begin with. When the traffic is not unexpectedly high the request is further processed as described by the 3G specification and dynamic bandwidth allocation takes care of further resource needs. The network should however have mechanisms to deal with unexpectedly high loads in parallel with operation of the bandwidth allocation mechanism. That is, the functions generated by a switching decision in one control step may simply be rejected by the CPU's load control algorithm after the bandwidth allocation step.

Now, the purpose of the algorithms that we will consider in section 3 is to accept as much load into the processor as possible without causing an overload on the CPU. The load is being primarily caused by traffic functions (see below), but also by functions on behalf of the higher network management layers (O&M) and other Operating System (OS) processes. Traffic functions are divided into connection management and mobility management functions. Here we will present some of the connection management functions. *Connection Setup* is used for establishing a signalling connection link to the UE. *Connection Release* ends the connection and releases the associated resources (channels, codes), *RAB (Radio Access Bearer) Establishment / Release* manage the connections to the UE used for service data transfer. To manage packet switched services different bandwidth channels are provided. Some of them can be shared by many connections, and others are dedicated. When the bandwidth of a channel does not suit a particular service a *Channel Switch* function is performed. *Paging* is performed when a UE is accessed form the Core Network. *Power Control* and *Admission Control* based on cell bandwidth are other function types. Mobility management functions are *Soft /Softer Handover*, which are performed to maintain a connection across cell boundaries and *Cell Update* to track the location of a UE.

To control the load on the processor we could either *reject or delay* some of the above-mentioned functions. A preliminary study of deferral methods shows that delaying traffic functions is not a solution [9]. Moreover many of these functions should not be subjected to rejections. Mobility functions have to be executed in order to maintain the connection status of the UE as it moves across cells. Rejecting basic control functions like Power Control would disturb the existing connections. Also disconnection from the network should not be refused. This leaves us with three candidates: Connection Setup, RAB Establishment and Channel Switches. Since RAB establishment is usually performed directly after a Connection Setup its control would not bring any benefit. Therefore in this paper we concentrate on two traffic function types: requests for new connections (R_1) and requests for channel switches (R_2). Rejecting a new connection is deemed less penalizing than dropping an established call, therefore the former may get a lower importance compared to the latter. In this paper we assume that all the other traffic tasks together with other OS tasks are directly sent to the processor and are therefore outside our control (thus referred to as direct tasks). The proposed

algorithms however, are not restricted to two task types or two importance levels.

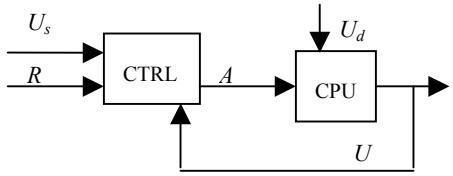


Figure 1. Schematic overview of processor protection

Cast as a simple control problem in its simplest form, we have the schematic diagram from Figure 1: The CPU to be protected from overload is our "plant" in a control terminology. The requests arriving are input to the controller (CTRL), which has the primary task of protecting the CPU from overloads. To do this it has access to the actual measured CPU utilisation, which is the sum of utilization caused by the number of accepted requests (denoted by A) and the utilisation from direct tasks (denoted by U_d). There are certain levels of desired utilisation, denoted here as a set value U_s. The reason for having a U_s that is different from 100% is the performance characteristics of the hardware and the risk of automatic resetting. That would jeopardise network availability during high load situations. Thus we want to keep U_s to stay at say 90%. Let C_i be estimated computation time for task type i. In order to estimate the utilisation caused by the mixture of arriving tasks ($R = R_1 + R_2$), we represent the required utilisation for each task type by c_i , where $c_i = C_i/P$ and P is the sampling interval. Thus, $U = U_d + \Sigma A_i c_i$, where $A = A_1 + A_2$ are the accepted tasks made up of types 1 and 2 respectively (new connections or channel switches). So far we differentiate only between task types. Next we add a user dependent QoS dimension. That is, we represent different user classes by the index j. R_{ij} will now represent the request type i from user type j. Then we can observe the network behaviour as the mix of user types and the mix of task types changes over time.

3. ALGORITHM DESIGN

The above parameters make several uncertainties explicit. We distinguish between two types of uncertainty: the resource needs of arriving requests and the variation over time with respect to user service policies. The first type manifests itself both in the mix of user requests and the mix of generated tasks (depending on whether the user is using voice, mail, SMS, web). Different user needs lead to different volumes of channel switches that in turn generate different volumes of tasks to be executed on the CPU. In this category of uncertainty we also find the volume of direct tasks arriving at the CPU (U_d above) and the claim on the processor with respect to each task (c_{ij}). The second type of uncertainty, namely the differentiation policy for user/task types, is treated at a higher level. It can be affected by the market and pricing information, or by configuration changes known at higher layers of network management. At the RNC level, we use variants of automatic control techniques to adjust for the first type, and deterministic algorithms to reflect the policy to deal with the second type. A change in the policy can thus be seen as the adjustment in the service level delivered to a particular user class over time.

This paper presents and compares two strategies. One approach combines feedback and feed forward control mechanisms with a deterministic pool allocation algorithm, while the other is based on a rejection-minimisation algorithm and state estimation. These two approaches are then compared with an existing leaky-bucket type load control algorithm. Figure 2 illustrates our architecture for the first approach, where the Load Control Unit (LCU) is instantiated by two boxes dealing with the two types of uncertainties mentioned above. Mechanisms for reflecting policy changes though represented as a (leftmost) box in our two architectures, is not the subject of this research in this paper and should be studied in presence of 3G business models and forthcoming data sets.

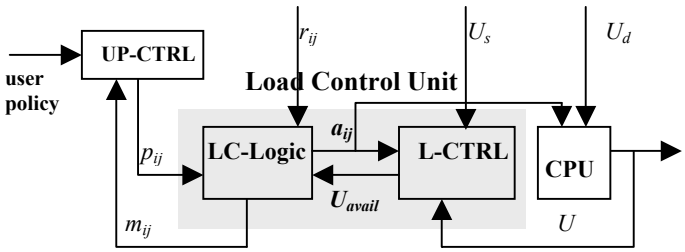


Figure 2. Schematic description of the first architecture

r_{ij}'s – are the numbers of incoming user requests of a given task type-i from user type-j.
p_{ij}'s – specify the percentage of the available utility which should be allocated to user requests of type (i,j).
a_{ij}'s – are the numbers of admitted requests of type (i,j).
m_{ij}'s – indicate the numbers of missed (rejected) requests of type (i,j).
U_{avail} – is the estimated available utilisation.
U – is measured (actual) utilization, which includes processing of admitted user tasks in addition to background tasks assigned by the operating system and the direct tasks not under control of the LCU.

3.1 First Architecture

The overall objective of the LCU is to achieve the stabilisation of the CPU utilisation around a set point U_s while managing individual user/task preferences according to p_{ij}'s. p_{ij} are set by a second controller (UP-CTRL) and represent the proportion of the available load that should be allocated to a certain request type (i,j) i.e. $\Sigma p_{ij} = 100\%$ of the available utilisation. This setting can be chosen based on task/user importance or other technical/economical data. The LCU is split into two different parts, a deterministic part and a feedback part. The first element (the LC-Logic) is responsible for accepting tasks based on a differentiation policy. This would be enough if our model were not subject to disturbances. That is, the c_{ij} for each task was accurate and the incoming direct tasks that are outside our control were 0. Since this is not true, we use the feedback from L-CTRL. This second element is used to adapt our current claim to the CPU using feedback about actually measured utilisation, and to follow the desired U_s. L-CTRL provides adjustments to variations in U_d and c_{ij}, and its output is the current available load U_{avail}. We have tested various P or PI controllers as instantiations of this unit (see section 5). In LC-Logic the utility shares decided by the UP-

CTRL will be used to compute the available load for each type: $u_{ij} = p_{ij} * U_{avail}$.

In reality the mix of incoming tasks may not correspond to the set points u_{ij}. If the requests of a certain type r_{ij} are demanding more utilisation from the processor than u_{ij} ($r_{ij} * c_{ij} > u_{ij}$), then we have a request excess for the type *ij*. Let's assume we allocate utilisation quotas of size u_{ij} to the incoming tasks. We will try to accommodate as many tasks as possible within their allocated quota. Then the following cases may arise:

1 We have a quota excess for all the types: $r_{ij} * c_{ij} > u_{ij}$. Then we accept for every *ij* as much as its quota can accommodate: $a_{ij} = u_{ij} / c_{ij}$

2 We have a quota shortfall for all types: $r_{ij} * c_{ij} < u_{ij}$. Then we can accept all incoming tasks: $a_{ij} = r_{ij}$.

3 We have a mix of excesses and shortfalls for different types *ij*. In this case, for each type we accommodate as much as possible within the allocated quota. If there is free space in some of the quotas, it will comprise an available pool. This pool will then be allocated to the remaining requests proportionally to the requested excess. Although this reallocation may not correspond to the initial priorities between tasks (priorities from which the initial quotas were established), it is a computationally efficient way to allocate all available CPU utilisation. In the end, if the pool is not big enough some of the tasks will have to be rejected.

Algorithm of LC-Logic

```
Pool := 0
Over := 0
u_ij = p_ij * U_avail
for i,j := 1 to N do
if (r_ij * c_ij <= u_ij) then
                a_ij := r_ij
                Pool := Pool + (u_ij - r_ij * c_ij)
                O_ij := 0
                m_ij := 0
        if (r_ij * c_ij > u_ij) then
                a_ij := u_ij / c_ij
                O_ij := r_ij * c_ij - u_ij
                Over := Over + O_ij
for i,j := 1 to N do
        if (O_ij > 0) then
                a_ij := a_ij + (Pool *O_ij /Over)/c_ij
                m_ij := r_ij - a_ij
```

3.2 Second Architecture

In the second approach all three elements of adaptation are rooted in control theory. The UP-CTRL is now replaced by a fuzzy controller that adaptively allocates the user-based p_j's as well as a weight for differentiating between task types w_j (see Figure 3).

The task of the constraint optimizer is to optimize the number of accepted tasks based on the following two criteria: 1) To accommodate as many received requests (r_{ij}'s) as possible, i.e. to ensure that a_{ij}'s are as close as possible to r_{ij}'s. 2) To ensure that the total computation time of the accepted requests does not exceed the pre-specified values of utilization. A cost function that attempts to have a_{ij}'s as close as possible to r_{ij}'s can be described as follows (where we consider a mix of two user types and two request types):

$$f(a_{11},a_{12},a_{21},a_{22}) = w_1(r_{11}-a_{11})^2 + (r_{21}-a_{21})^2 + w_2(r_{12}-a_{12})^2 + (r_{22}-a_{22})^2 \quad (1)$$

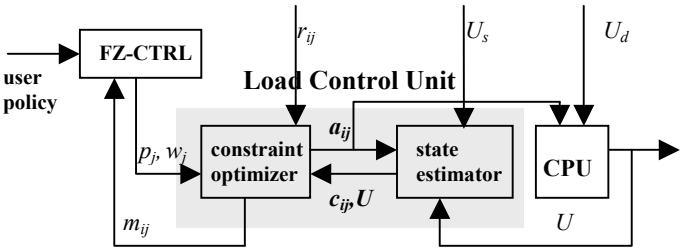


Figure 3. Schematic description of the second architecture

In the above definition, w_1 and w_2 are the weights that represent the priorities of type 1 requests (e.g. channel switches) over type 2 (e.g. new connections). The cost function of equation (1) does not take into consideration the pre-specified constraints on the CPU utilization:

$$\begin{aligned} a_{11}c_{11} + a_{21}c_{21} &= U_1 \\ a_{12}c_{12} + a_{22}c_{22} &= U_2 \end{aligned} \quad (2)$$

U_1 and U_2 at each sample time are calculated based on pre-specified percentages of the measured CPU utilization U in the previous sample, i.e. U_1 and U_2 at sample "k" can be specified as:

$$\begin{aligned} U_1(k) &= p_1 \times U(k-1) \\ U_2(k) &= p_2 \times U(k-1) \end{aligned} \quad (3)$$

where p_j describe the share of each user in the available bandwidth. In order to incorporate the constraints of equations (2) in a more general cost function, the method of Lagrange Multipliers for optimization can be used, i.e. the new cost function can be defined as:

$$\begin{aligned} f(a_{11},a_{12},a_{21},a_{22},\lambda_1,\lambda_2) = w_1(r_{11}-a_{11})^2 + (r_{21}-a_{21})^2 + w_2(r_{12}-a_{12})^2 + \\ + (r_{22}-a_{22})^2 + \lambda_1(a_{11}c_{11}+a_{21}c_{21}-U_1) + \lambda_2(a_{12}c_{12}+a_{22}c_{22}-U_2) \end{aligned} \quad (4)$$

In order to optimize the above cost function, the following set of equations must be solved:

$$\frac{\partial f}{\partial a_{11}} = 0, \frac{\partial f}{\partial a_{21}} = 0, \frac{\partial f}{\partial a_{12}} = 0, \frac{\partial f}{\partial a_{22}} = 0, \frac{\partial f}{\partial \lambda_1} = 0, \frac{\partial f}{\partial \lambda_2} = 0 \quad (5)$$

Calculating the derivatives, equations of (5) will result in the following set of linear equations:

$$\begin{aligned} \frac{\partial f}{\partial a_{11}} &= 2w_1r_{11} - 2w_1a_{11} + \lambda_1c_{11} = 0 \\ \frac{\partial f}{\partial a_{21}} &= 2r_{21} - 2a_{21} + \lambda_1c_{21} = 0 \\ \frac{\partial f}{\partial a_{12}} &= 2w_2r_{12} - 2w_2a_{12} + \lambda_2c_{12} = 0 \\ \frac{\partial f}{\partial a_{22}} &= 2r_{22} - 2a_{22} + \lambda_2c_{22} = 0 \\ \frac{\partial f}{\partial \lambda_1} &= a_{11}c_{11} + a_{21}c_{21} - U_1 = 0 \\ \frac{\partial f}{\partial \lambda_2} &= a_{12}c_{12} + a_{22}c_{22} - U_2 = 0 \end{aligned} \quad (6)$$

Solving the above set of equations results in the following task acceptance criteria:

$$a_{11} = r_{11} - \frac{c_{11}(r_{11}c_{11} + r_{21}c_{21} - U_1)}{c_{11}^2 + w_1 c_{21}^2} \quad (7)$$

$$a_{21} = r_{21} - \frac{w_1 c_{21}(r_{11}c_{11} + r_{21}c_{21} - U_1)}{c_{11}^2 + w_1 c_{21}^2}$$

$$a_{12} = r_{12} - \frac{w_2 c_{12}(r_{12}c_{12} + r_{22}c_{22} - U_2)}{c_{22}^2 + w_2 c_{12}^2}$$

$$a_{22} = r_{22} - \frac{c_{22}(r_{12}c_{12} + r_{22}c_{22} - U_2)}{c_{22}^2 + w_2 c_{12}^2}$$

The following two observations can be made. First, the set of equations in (7) can be used in any sample point k to update the values of accepted number of tasks based on r_{ij} and c_{ij}, and the allocated utilization for the user j. Second, if $r_{11}c_{11} + r_{21}c_{21} = U$, then using the above equations: $a_{11} = r_{11}$, which gives the intuitive and expected strategy of accepting all requests. Also, we should ensure that the computed a_{ij} follow this constraint: $0 < a_{ij} \leq U_j / c_{ij}$.

Note that so far c_{ij} has been a constant representing the estimated utilisation per task and U_d has been null. In order to have much more accurate estimates for c_{ij}'s and U_d, we should use a Kalman estimator (that is an optimal estimator). This estimator, at each sample point "k", uses the measured values of the total amount of utilization for the past few samples (i.e. $U(k-p), U(k-p-1),...,U(k)$), and the number of accepted tasks in the past few samples (i.e. $a_{ij}(k-p), a_{ij}(k-p),..., a_{ij}(k)$) to find the optimal estimation of c_{ij}'s and U_d for the next sample "k+1". In this approach the relation between c_{ij}'s, U_d and U at sample "k" can be describes as:

$$U(k) = a_{11}(k)c_{11}(k) + a_{12}(k)c_{12}(k) + a_{21}(k)c_{21}(k) + a_{22}(k)c_{22}(k) + U_d(k) \quad (8)$$

We expect that by using the optimally estimated values of the computation times, the resulting controller can avoid under-utilisation more successfully.

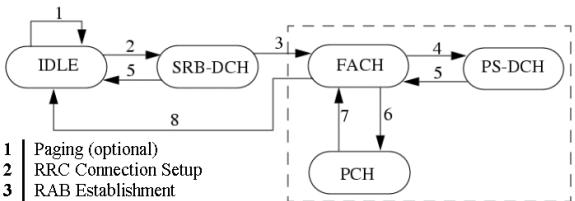
The FZ-CTRL in Figure 3 controller uses the rate of rejections in any of the task categories. By considering these values as input, the controller decides about re-proportioning of utilization time, i.e. updating the ratios p_j. A set of fuzzy rules can best implement such a controller that would also adjust the relative priority between task types (w_j). These rules are generated based on the heuristic knowledge of the experts in the field and then optimized using a set of training data.

4. EVALUATION OF ARCHITECTURES

In order to evaluate our algorithms in a realistic setting we have implemented a simulation environment in which traffic models for various user scenarios can be created and tested against the designed algorithms. The simulation environment enables us to test-run the scenarios on an existing simulated model of the target machine together with the relevant operating system and middle-ware.

The traffic models generated are derived from available data, providing different RNC traffic functions for different types of user applications like voice, mail, SMS and web browsing. The models also reflect user behavioural patterns with respect to available load, such that an increase in the service levels (reflected in fewer rejections) or vice versa lead to different future developments in the traffic scenario. As an example we will show how we generate a "web browsing" session. Figure 4 shows the states in which a connection can be and the possible transitions, which result in different traffic functions.

The simulation platform provides an excellent environment in which many parameters in the designed RNC algorithms can be varied and experimented with, e.g. various P, PI controllers, different coefficients, different sampling intervals, different p_{ij} levels. However, it should be noted that all elements of user-differentiation introduced here are for research purposes and constitute exploratory ideas. More precise models can only be derived when service and business models are in place.



1	Paging (optional)
2	RRC Connection Setup
3	RAB Establishment
4	Channel Switch (transfer data)
5	Channel Switch
6	Channel Switch
7	Channel Switch
8	RRC Connection Release

WWW Browsing parameters	Distribution
Intensity (sessions/second/CPU)	Poisson
Time in DCH per WWW request	Neg. exponential
Number of www requests	Geometric
Delay between www requests	Neg. exponential

Figure 4. The "www browsing" session

The results we show below concentrate on overload scenarios. The offered load quickly rises to about 30% over the maximum capacity of the processor. The load profiles provide us with a realistic enough approximation according to our knowledge about 3G traffic behaviour, and are obviously open to further experiments. In addition, in order to check the exact behaviour of the algorithms under controlled experiments (which may never the less be far from realistic traffic scenarios), we have tried several artificial but carefully designed scenarios too. These scenarios were used to confirm our pre-simulation predictions and intuitions about how the algorithms would behave in a particular setting. In the following section we provide some selected results and comparisons.

5. SIMULATION RESULTS

We present a comparison of the two architectures presented above with an existing algorithm at Ericsson, which uses the leaky bucket mechanism. On a sample-by-sample basis, the core difference between the existing and the new algorithms is in terms of the LCU design. In the following experiments we therefore exclude the effect of a long-term adjustment of the system to user policy revisions (UP-CTRL or FZ-CTRL). Though a Kalman estimator would be optimal for dealing with uncertainties, our experience shows good results even with a few simple estimators; e.g. in the following experiments we used a simplified feedback mechanism. When the system is overloaded (underloaded) we decrease (increase) U_{avail} ($U_{avail} = U_s - U_d$) with a constant step.

The leaky bucket type algorithm, used to control the load on the processor, works in the following manner: Whenever a task arrives, first it checks whether enough space is available in the bucket. For each task type a different rejection level can be specified. If the "water" in the bucket is above the specified level the task will be rejected. If it is below, the task is accepted and an amount of "water" corresponding to the execution time of the task is poured into the bucket. At regular intervals, "water" is taken from the bucket, to emulate a constant outflow. To provide adaptability a heuristic feedback mechanism is used. The amount of water to put into the bucket is raised or lowered if the average measured load on the processor is lower or higher than U_s.

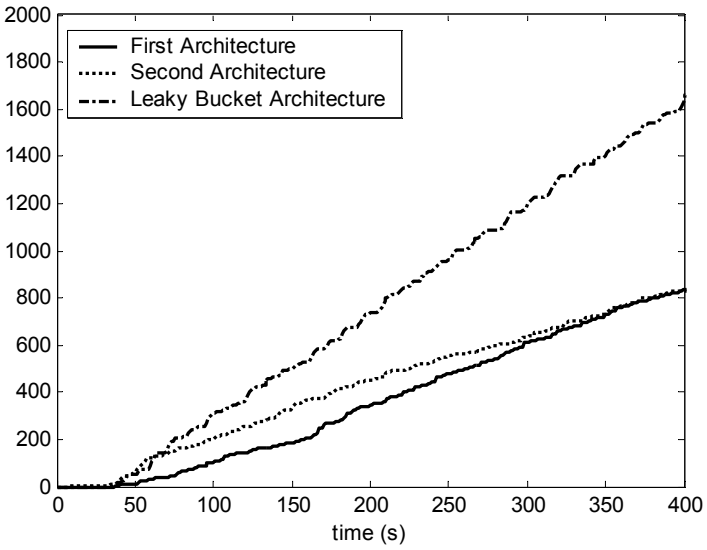


Figure 5. Cumulative overload beyond the desired threshold

Recall that our load control system has two goals. First, to avoid reaching CPU load levels beyond a particular threshold, and second, to differentiate between the user/task types as specified by a given policy. Figure 5 shows how our two architectures compare to the base line algorithm in the context of a realistic traffic scenario. The Y-axis shows the accumulated overload beyond the given threshold (U_s), expressed in terms of percentages. This graph shows that the leaky-bucket algorithm has a steeper rate with respect to violating the "no overload" requirement.

In order to better explain Figure 5, Figure 6 shows the load on the processor generated by the leaky bucket architecture compared with our first architecture (Pool + P-controller). The amount over the 80% line is the overload presented in the previous figure. Figure 6 also shows that the P-controller is quicker to adapt to the changes in U_d, which increases the overload protection of the processor. On the other hand, the leaky bucket has slightly less rejections than our architectures. We argue that this is because it also allows more load on the processor. The second architecture presents very similar load results compared to the first architecture. Our experiments show that no matter which differentiation policy is used, the use of feed back mechanisms ensures a more stable behaviour in the light of inaccuracies in the c_i and U_d estimates.

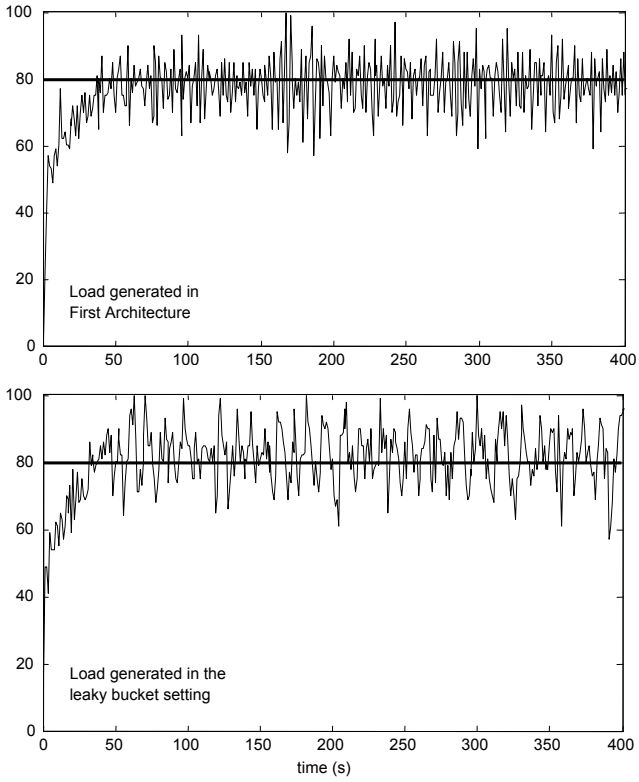


Figure 6. Load comparison

Let's now look at the differentiating behaviour of the two proposed algorithms. These algorithms are intrinsically different in the following sense: Pool allocation is useful when the differentiation policy can be naturally described as shares of the CPU, whereas the constraint optimiser is beneficial if the proportion of rejections is in focus. This relative prioritising is illustrated for the 50/50 case in Figure 7 where the number of accepted and rejected tasks is plotted for one user type. With a 50/50 we expect half of the CPU share to go to new connections and half to channel switches. In this case the new connections are not enough to occupy the entire allocated share, thus the pool is reallocated to the channel switches. This results in the number of accepted channel switches to be above the 50% share. We would expect that the number of rejected new connections to be zero, but there are few such rejections due to estimation errors of the adaptive part. In the second architecture, since both types have the same priority (50/50), the constraint optimiser tries to reject as many new connections as channel switches.

Our next set of experiments was designed to separate the effect of the two boxes in each strategy: the adaptive part that deals with uncertainties in c_{ij} and U_d, and the deterministic algorithms. These experiments showed that the U_d, being at times very oscillative, has a stronger impact on the results than the c_{ij}. However, the experiments show that both adaptation elements (feed back controller or state estimator) can be tuned with the right choice of coefficients. Illustrations are omitted due to space restrictions.

Our final illustration is a proof of concept for the multi-user multi-task differentiation (Figure 8). This is especially interesting when the offered load has a different mixture compared to the expected load. We illustrate the workings of the constraint

optimiser in detail. We have chosen a scenario with w_j corresponding to 1000/1 (almost always prefer channel switches) and p_j's corresponding to 80/20 (allocate ~80% of the available load to user one and ~20% to user 2). The offered load happens to have a 50/50 mix in tasks generated by user 1 and user 2.

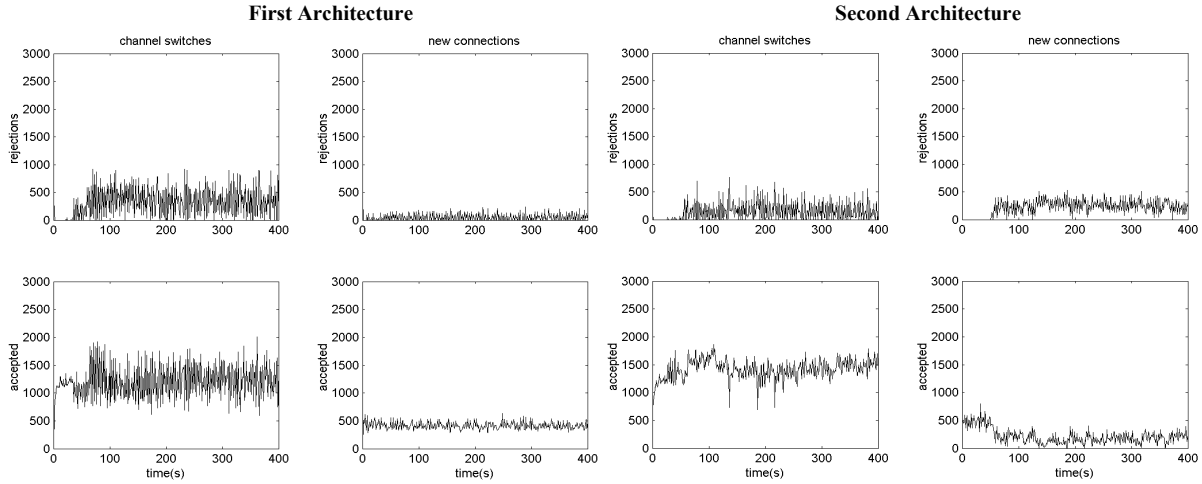


Figure 7. Behaviour with a 50/50 differentiation policy

The top row shows the percentage of the available load ($U_s - U_d$) occupied by user 1 and user 2 respectively. To begin with, none of the users fill up their allocated share, so the mix is different from the 80/20 proportions. When the system is at overload (around 130 seconds) the accepted load corresponds to the allocated quota. The next two rows show the load in the first row divided between channel switches and new connections for each user. For user 1 the allocated quota is enough to accommodate new connections (despite 1000/1 priority to channel switches). Since user 2 has only a 20% share, it gets completely occupied by channel switches and almost entirely rejects new connections, which is the expected behaviour.

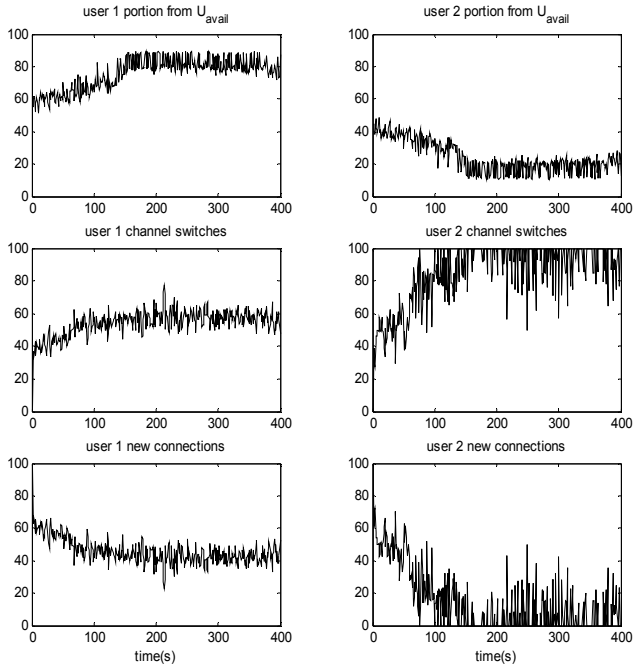


Figure 8: Multi-user scenario with unbalanced load

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented adaptive architectures for load control in an RNC node of a 3G network. The main goal was to provide a protection scheme for the processor while rejecting as few control tasks as possible, and ensuring that in case of overload, different task types are accepted based on a policy decided by the service provider. For example, it would be a strange situation if a user that was accepted into the system based on certain QoS parameters and bandwidth availability, was later rejected in a load control algorithm. System behaviour under (partial) traffic overloads should be consistent with the provider policy. The presented architectures can be partitioned in two main components. The adaptive part is a feedback-feedforward controller, whose function is to deal with the uncertainties in our model e.g. the execution time of tasks and the load on the processor that is outside our control. The deterministic part is the differentiating algorithm, which ensures that the available load is partitioned corresponding to a certain policy. The presented architectures successfully protect the processor from overload, and also enforce the specified QoS.

We have compared our proposed architectures with an existing regime that is based on the Leaky bucket. While this algorithm is quite satisfactory in the current settings (no user differentiation), it enforces absolute priorities among task types. If the need for relative prioritisation arises in 3G business models, then absolute priorities may not suffice. Our architectures formulate QoS requirements as relative priorities between different task/user types, and enforce them by share/rejections quotas for the different types. The algorithms have been evaluated in the context of realistic traffic scenarios using our implemented simulator and traffic generator. While the Pool is straightforward to implement, it is not suited to implement strict priorities among tasks, due to the pool reallocation mechanism. The optimiser on the other hand, does not enforce a proportional share of the CPU since it is based

on rejection minimisation. The analysis of both architectures enables the service provider to focus on the technique most suited in a particular setting.

The next issue was the adaptive character of the architectures. Confronted with a strong oscillating U_d neither the P-controller nor the leaky bucket estimator show an ideal behaviour. The P-controller has the drawback that it follows the oscillative character of the U_d, one control interval behind. On the other hand the averaging estimator used in the leaky bucket architecture is at times too slow to react, leading to the possibility of overload of the processor. The simple state estimator falls somewhere in the middle. To improve the behaviour of our control schemes it would be necessary to have a better estimation of the U_d. This entails treating traffic functions and management tasks differently (e.g. delaying management tasks selectively). In particular, there are tasks whose rejection increases the load (or at least does not decrease it since it generates other tasks). The rejection of these could be taken account of in estimation of future U_d.

There are several directions for extending the work after the initial analysis of the algorithms. Further studies using variants of the UP-CNTL and the FZ-CTRL is dependent on realistic user and business models. The formal analysis of each algorithm is also interesting: e.g. how to extend the constraint optimiser solution when the number of users/task types increases beyond 2. Consideration of the load control algorithms´ overheads (themselves running on the same CPU) is also worth studying. There is obviously a limit (beyond two user types) after which the optimiser itself will have a high execution overhead. Studying interdependencies between CPU load control and other resource allocation problems (e.g. bandwidth) is another interesting track of work. The experiments so far show that a load control algorithm can be faced with such overload situations that even if it works perfectly well, there are still no tasks accepted. Multi-processor load balancing is a way to treat such overloads.

7. ACKNOWLEDGEMENTS

The authors wish to thank Pär Gustavsson and Sören Holmström at Ericsson for discussions and feedback during this work.

8. REFERENCES

[1] Abeni, L., and Buttazzo, G. Hierarchical QoS Management for Time Sensitive Applications. Proceedings of the IEEE Real-Time Technology and Applications Symposium, May 2001.

[2] Atiquzzaman, M., and Hassan, M. (Editors). Quality of Service over Next-Generation Data Networks. The Convergence of Information Technologies and Communication (ITCOM), Denver, Colorado, August 2001.

[3] Aurrecoechea, C., Campbell, A.T., and Hauw, L. A Survey of QoS Architectures. ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture, Vol.6 No.3, pg. 138-151, May 1998.

[4] Choi, S., and Shin, K. G. A Comparative Study of Bandwidth Reservation and Admission Control Schemes in QoS-sensitive Cellular Networks. Wireless Networks, 2000, vol. 6: pp 289-305.

[5] Dahlberg, T. A., and Jung, J. Survivable Load Sharing Protocols: A Simulation Study. Wireless Networks, May 2001, Vol. 7, pp. 283-296.

[6] Dahlberg, T. A., and Subramanian, K. R. Visualization of Mobile Network Simulations. Simulation, Journal of the Society for Computer Simulation International, to appear.

[7] El-Kadi, M., Olariu S., and Abdel-Wahab, H. A Rate-Based Borrowing Scheme for QoS Provisioning in Multimedia Wireless Networks. IEEE Transactions on Parallel and Distributed Systems, February 2002, vol. 13, no. 2: pp. 156-167.

[8] Liao, R., and Campbell, A. A Utility-Based Approach for Quantitative Adaptation in Wireless Packet Networks. Wireless Networks, September 2001, vol. 7: pp 5 41-557.

[9] Lingvall, T. Load Control in a Radio Network Controller within UMTS. Linköping University Masters Thesis, LiTH-IDA-Ex-02/X, March 2002.

[10] Mahadevan, I., and Sivalingam, K. Architecture and Experimental Framework for Supporting QoS in Wireless Networks Using Differentiated Services. Mobile Networks and Applications, August 2001, vol. 6: pp. 385-395.

[11] Oliveira, C., Kim, J. B., and Suda, T. An Adaptive Bandwidth Reservation Scheme for High-Speed Multimedia Wireless Networks. IEEE Journal on Selected Areas in Communications, August 1998, vol. 16, no. 6: pp. 858-874.

[12] Richardson, P., Sieh, L., and Ganz, A. Quality of Service Support for Multimedia Applications in Third Generation Mobile Networks Using Adaptive Scheduling. Real-Time Systems, November 2001, 21(3): pp. 269-284.

[13] Stankovic, J.A., He, T., Abdelzaher, T.F., Marley, M., Tao, G., Son, S.H., and Lu, C. Feedback Control Real-Time Scheduling in Distributed Real-Time Systems. IEEE Real Time Systems Symposium, December 2001.

[14] Staehle, D., Leibnitz, K., and Tsipotis, K. QoS of Internet Access with GPRS. Wireless Networks, 2002, to appear.

[15] Wang, S., Wang, Y.-C., and Lin, K.-J. Integrating Priority with Share in the Priority-Based Weighted Fair Queuing Scheduler for Real-Time Networks. Real-Time Systems, March 2002, 22(1-2): pp. 119-149.