PERFORMANCE ANALYSIS OF THE AP AND THE SSC AND DESIGN OF THE SYSTEM OVERLOAD CONTROLS OF INDIA'S C-DOT DIGITAL SWITCHING SYSTEM

B. Madhavi and Anurag Kumar
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore, 560 012, INDIA
anurag@vigyan.ernet.in

Abstract

Government of India's Centre for Development of Telematics (C-DoT) has developed a modular, distributed telephone switching system. There is an ongoing effort to model, analyse, and optimise the teletraffic performance of this switching system. In this paper we report the methodology and results of a part of this work. We use analytical queuing models to estimate the capacities of the Administrative Processor, and the Space Switch Controller. We formulate the system overload control problem as a constrained optimisation problem, solve the problem, and use simulations to evaluate various adaptive overload control algorithms.

1 Introduction

Government of India's Centre for Development of Telematics(C-DoT) has developed a modular, distributed, digital switching system that will eventually be capable of supporting upto 16,000 ports and an attempt rate of 300,000 calls/hour (see Figure 1). There has been an ongoing effort to analyse and optimise the performance of the various processor based modules of this switching system. A detailed simulation model was used to estimate the capacity of a Base Module to design an overload control for it, and to show how its capacity can be improved. This work was reported in [2]. In this paper we report on the methodology we have used to (i) analyse the performance of two processor based modules, namely the Administrative Processor (AP) and the Space Switch Controller (SSC), and (ii) design the system level overload controls.

The AP and the SSC perform call processing tasks and also participate in various lower priority activities such as testing and audits. We identify the various performance criteria that mnst be met for each of these classes of tasks in each of the two modules. We then develop a queueing model, with three priority levels, that applies to both the modules. The analysis of this queueing model reduces to that of a standard "vacation model". Numerical computations from well known Laplace transform formulae yield the desired performance measures. We provide numerical results based on preliminary real time measurements obtained from C-DoT.

We then turn to the problem of designing the system level overload control, i.e., an overload control strategy that protects all the call processing modules. At variance with the ad hoc techniques that have previously been adopted for such problems, we formulate the problem as one of minimising average call set-up time while keeping throughput as large as possible. This yields a contrained nonlinear minimisation problem. The problem is easily solved and a scheme is developed for on-line tuning of the controls (i.e., an adaptive overload control). Simulations are used to study the performance of the algorithm. Preliminary results show that the algorithm gives good transient and steady state performance.

This paper is organised as follows In Section 2 we give an overview of the C-DoT DSS architecture. In Sections 3 and 4 we describe the performance analysis of SSC and AP. In Section 5 we briefly present the methodology for designing the system level overload controls.

This paper is a brief summary of the methodology and results. The details are available elsewhere. See [2], [7], [6].

2 Overview of C-DoT DSS Architecture

The C-DoT DSS has a modular and distributed architecture (Figure 1) consisting of a number of processor based modules, namely an Administrative Module (AM), a Central Module (CM), and a number of Base Modules (BM).

The AM provides the functions of overall system initialisation, systemwide maintenance, *call routing and trunk allocation*. The system administration consoles are connected to the Input-Output Processor (IOP), which interfaces to the system ria the AM.

The CM consists of a Space Switch (SS), the Space Switch Controller (SSC), and a Central Message Switch (CMS). The space switch provides the digital paths for voice connections between BMs. The SSC ***allocates and deallocates inter-BM speech paths*** through the space switch. The CMS is basically a simple packet switch, and is connected to the BMs and the SSC by 64 kbps links and to the AM by a 128 kbps link. The various modules coxnmuaicate with each other by sending messages that are switched between the links by the CMS. A simple stop-and-wait ARQ [8] protocol is used on each link. End-to-end message integrity is maintained by an end-to-end go-back-n ARQ protocol [8].

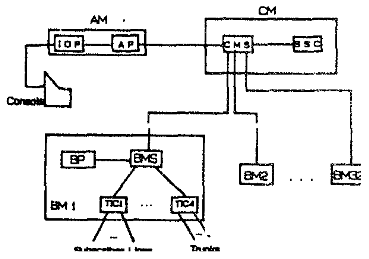

Figure 1: Architecture of the C-DoT DSS

Lines and trunks terminate on the Terminal Interface Controllers (TICs) in the Base Module. Each Base Module comprises four TICs, each providing access to 128 subscriber lines or trunks. Hence the maximum configuration of 32 BMs supports 16,000 lines/trunks. A TIC performs fnnctions such as origination detection, digit reception, answer detection, digit outpuking etc., for the line and trunk terminations. A BM also contains a Time Switch (TS) (time slot interchanger), a Time Switch Controller (TSC), a Base Message Switch (BMS) and a Base Processor (BP). The time switch is part of the overall time-space-time switching fabric, and is also used for swiiching intra-BM calls. The BMS provides message switching between the TICs and the BP. The BP processes call stimuli in the BM.

The processor modules run the CDOS operating system. This is a multitasking operating system, in which processes interact by passing messages to each other. The setup of a call involves the interaction of several software processes running in the processors of the various modules. Each of these processes can either be eternal or dynamic. Eternal processes are created during system initialization, whereas dynamic processes are created on demand by the operating system and they terminate on completion of their task (typically on the completion of a call). The processes communicate between themselves through messages. As mentioned above, when the processes are on different modules, the interprccessor messages are switched by the CMS.

Each process has associated with it a message queue. When a process gets its turn to run on the processor, it processes a message in its message queue. Upon completion of this work, the process joins the back of a queue of processes waiting for service at the processor. The operating system permits process priorities, but the priority is nonpreemptive. Further, the queue of messages waiting at a process may also he ordered by message priorities. Note here that, although priorities between software processes are nonpreemptive, hardware priority is preemptive. Thus, for example, an interrupt by a link controller will interrupt a software process. This has important implications for central resources (e.g., AP and SSC) that must communicate frequently with all the other modules.

The details of a typical call set up are given in [2].

3 Performance Analysis of the Space Switch Controller

3.1 The SSC and its activities

Recall from Section 2 that the Central Module houses the Space Switch which is a part of the overall time-space-time switching fabric and is used for switching inter-BM calls. The SSC is concerned with establishing and tearing down paths through the Space Switch. This is the only call setup related activity of the SSC.

The SSC communicates with the various module processors through mes sage transmissions and receptions that cause hardware interrupts to the SSC. Each link interrupt needs service from an interrupt service routine that is exe-'cuted by the SSC processor. The protocol used for message exchange between processors is such that, each message results in 4 interrupts to a processor. A message reception/transmission is followed by its link acknowledgement, then by an end-bend acknowledgement and finally by a link acknowledgement for the end-to-end acknowledgement. Recall that hardware interrupts are served at the highest priority and are preemptive.

In the current software version, some maintenance and audit processes are run in the SSC. The SSC periodically receives sanity monitor punches (SMPs) from the various processors, namely the BPs, the AP, the CMS. These are served at a higher priority khan calls. In addition, the SSC performs tests that are served at alower priority than call processing tasks. There are three types of tests: (i) Sequential Memory Access test, (ii) Voice Path test, and (iii) Link test. Lastly the SSC is required to perform auditing functions that are also lower in priority than call processing. One such audit function is the detection and release of hanging buffers.

3.2 SSC processor occupancy as a function of load

We have seen from the previous section that tests and audits are performed only when the processor is not involved in the processing of calls, sanity punches or interrupts. Such work is said to be deferrable. There are three non deferrable tasks in the SSC, namely, processing of calls, processing of sanity punches and processing of interrupts. We shall refer to the fraction of time that the SSC is busy performing non-deferrable tasks as SSC occupancy.

Let

ρ = SSC processor occupancy (due to non deferrable work)

λ = Busy Hour Call Attempt (BHCA) rate

σ = Fraction of attempts at the BMs that arc good attempts (i.e., not partial-dials, no-dials, etc.)

λ_C = Arrival rate of requests for path setup and tear down = $2\sigma\lambda$

b_{setup} = Service time of a path set up request

$b_{teardown}$ = Service time of a path tear down

λ_I = Arrival rate of link interrupts to the SSC processor

b_{int} = Interrupt service time

ρ_{SMP} = Occupancy of the SSC processor due to sanity monitor punches (about 1-2%)

We note that b_{setup}, $b_{teardown}$ and b_{int} are modelled as fixed times and are obtained from measurements. It follows that the SSC processor occupancy is given bv

$$\rho = \rho_{SMP} + \sigma\lambda(b_{setup} + b_{teardown}) + \lambda_I b_{int} \quad (1)$$

Note that the interrupt arrival rate λ_I comprises the interrupts related to calls and also the interrupts due to other messages. Recall that the message protocol is such that each message (sent or received) causes 4 link interrupts. Each call generates 3 messages at the SSC: a switch path request from the AP, a termination message from the SSC to the destination BP, and a switch path release message to the SSC.

Letting

λ_{I_0} = interrupt rate due to non-call related messages

we have,

$$\lambda_I = 12\sigma\lambda + \lambda_{I_0}, \quad (2)$$

λ_{I_0} is easily calculated by accounting for the message traffic due to sanity monitor punches, and the various tests. It turns out that, for a 32 BM configuration, and for the current software version, λ_{I_0} = 150 per second. For details see [7].

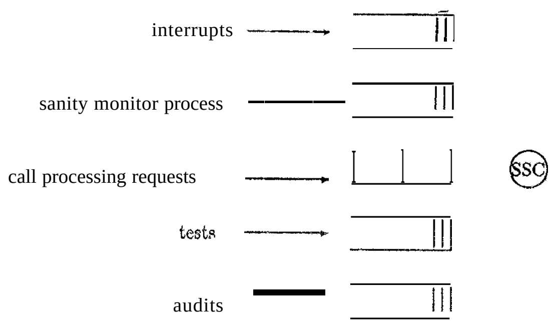

Figure 2: Five priority queuing model for the SSC

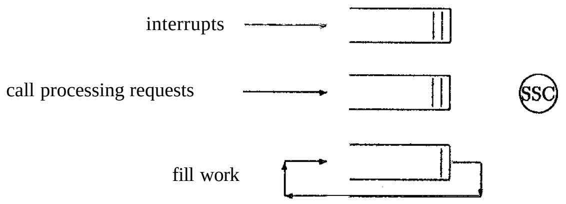

Figure 3: Three priority queuing model for the SSC

3.3 Queuing model for the SSC

From Section 3.1, it is clear that the SSC processor has to serve 5 types of customers at different priority levels. This would yield a model as shown in Figure 2. The SSC occupancy due to sanity monitor process is of the order of 1-2%. We remove the SMP out of the model by inflating all service times by a factor of 1/0.98 or 1/0.99.

Note that tests and audits are performed at the lowest priority, but are not indefinitely deferrable. There is an attempt to perform tests and audits in segments of 10 ms. approximately. Though this traffic is not always present, we model it as though it is always queued for service at the SSC and call it "fill work". It then becomes a point of interest to know the exact number of till work segments (of 10 ms. each) that can be served by the SSC in a given time.

Thus we have a 3-priority queue model consisting of interrupts, path setups/tear downs and fill work segments as shown in Figure 3.

Note that interrupts can occur either during a call service time or the service time of a fill work segment. These interrupts receive preemptive service at the highest priority. This suggests that we can have a further simplified 2-priority model consisting of calls and fill work segments both of whose service times are modified appropriately to take into account the effect of the preempting interrupts.

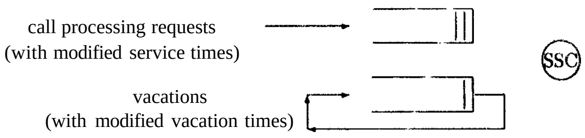

Figure 4: Two priority queuing model for the SSC

Moreover, since fill work segments are performed only in the absence of calls we can model these as nonpreemptible "vacations" taken by the server whenever the call queue is empty.

If λ_C, the arrival rate of path setups and teardowns into the SSC, is assumed to be Poisson, then the above modification of service times due to interrupts can be done exactly, and we have the standard M/G/1 queue with vacations as the model for SSC [4]. This final model is shown in Figure 4.

3.3 Performance criteria

We first note that the call sojourn times at the SSC directly add up to the end-to-end set up delay. Hence an important performance measure is switch path request sojourn time at the SSC.

Though tests are deferrable, there is a requirement that they cannot be delayed by more than a specified amount of time. It follows from our 2-priority queue model that this requirement will be met if there is a limit on the lengths of call busy periods. Similarly, a certain number of audits must be completed within a specified amount of time. Thus completion time for a cycle of tests and audits is another performance measure of interest.

These performance measures generate the following performance requiremeuts, where t_1, t_2 and t_3 are specified time constraints, and α_1, α_2, α_3 are probabilities (close to 1) with which these constraints must be met.

$$P(\textit{Call setup or teardown time} < t_1) > \alpha_1 \tag{3}$$

$$P(\textit{Call busy period} < t_2) > \alpha_2 \tag{4}$$

$$P(\textit{Cycle time for F fill work segments} < t_3) > \alpha_3 \tag{5}$$

We see that the performance requirements are specified in terms of probability distributions of the corresponding random variables. Hence we resort to the determination of Laplace Stieltjes Transforms [4] for the distributions of

1. Sojourn times in the call processing queue
2. Busy periods in the call processing queue
3. Completion time for F fill work segments

These are then numerically inverted using a numerical technique [3] to obtain the probability measures.

3.4 Analysis of the queuing model

3.4.1 Modification of call service times and fill work segments to incorporate interrupts

Let X denote the call service time random variable, Y the interrupt service time random variable, and V the vacation time random variable. Thus, for the workload model in Section 3.2,

$$X = \begin{cases} b_{setup} & \text{with probability } 1/2 \\ b_{teardown} & \text{with probability } 1/2 \end{cases}$$

Further, $Y = b_{int}$ with probability 1. Letting

b_{fill} = fill work segment length (= 10 msec in the current system)

we have $V = b_{fill}$ with probability 1.

Proceeding more generally, however, let $\tilde{x}(s)$ denote the Laplace Stieltjes Transform (L.S.T) of the cumulative distribution function (c.d.f.) $X(t)$ of X. Let X_I denote the call service time modified bv hardware interrupts, and define $X_I(t)$ and $\tilde{x}_I(s)$ analogously. Similarly, define $Y(t), \tilde{y}(s)$ and $V(t), \tilde{v}(s), V_I, V_I(t), \tilde{v}$

Assuming that the interrupts arrive in a Poisson process independent of the call arrival process, we now proceed to find $\tilde{x}_I(s)$.

Since the interrupts receive preemptive service at the highest priority, the interrupt service queue is just an M/G/1 queue. Let G_I denote its busy period distribution, and $\tilde{g}_I(s)$ the L.S.T. of G_I [4]. It follows that

$$\tilde{g}_I(s) = \tilde{y}(s + \lambda_I - \lambda_I \tilde{g}_I(s))$$

Conditioning on the length of the unmodified call service time X , we get,

$$\begin{aligned} \tilde{x}_I(s) &= \int_0^\infty e^{-sx} \sum_{k=0}^{\infty} \frac{(\lambda_I x)^k}{k!} e^{-\lambda_I x} (\tilde{g}_I(s))^k \, dX(x) \\ &= \int_0^\infty e^{-x(s+\lambda_I(1-\tilde{g}_I(s)))} \, dX(x) \end{aligned}$$

i.e.,

$$\tilde{x}_I(s) = \tilde{x}(s + \lambda_I(1 - \tilde{g}_I(s))) \tag{6}$$

Similarly, we have

$$\tilde{v}_I(s) = \tilde{v}(s + \lambda_I(1 - \tilde{g}_I(s))) \tag{7}$$

For the work load model discussed above,

$$\begin{aligned} \tilde{x}(s) &= 0.5e^{-sb_{setup}} + 0.5e^{-sb_{teardown}} \\ \tilde{y}(s) &= e^{-sb_{int}} \\ \tilde{v}(s) &= e^{-sb_{fill}} \end{aligned}$$

Also it follows that,

$$EX_I = EX + EX \frac{\lambda_I b_{int}}{1 - \lambda_I b_{int}} = \frac{(b_{setup} + b_{teardown})/2}{1 - \lambda_I b_{int}} \tag{8}$$

$$EV_I = EV + EV \frac{\lambda_I b_{int}}{1 - \lambda_I b_{int}} = \frac{b_{fill}}{1 - \lambda_I b_{int}} \tag{9}$$

3.4.2 Call sojourn times

Now we have an M/G/1 vacation model for the SSC with arrival rate λ_C, service time L.S.T. $\tilde{x}_I(s)$ and vacation time L.S.T. $\tilde{v}_I(s)$. Let T_C be the random variable for the sojourn time of a call at the SSC. Let $\tilde{t}_C(s)$ denote the L.S.T. of the c.d.f.. $T_C(t)$ of T_C. Let T_{C_0} and $\tilde{t}_{C_0}(s)$ denote the random variable and its L.S.T. for call sojourn time in an M/G/1 queue without vacations. It follows from the Pollaczek - Khinchine formula [4] for sojourn time in an M/G/1 queue (without vacations), that

$$\tilde{t}_{C_0}(s) = \frac{s(1 - \rho_{CI})\tilde{x}_I(s)}{s + \lambda_C - \lambda_C \tilde{x}_I(s)} \tag{10}$$

where,

$$\rho_{CI} = \lambda_C E(X_I) \tag{11}$$

Clearly ρ_{CI} is the processor occupancy due to calls and the interrupts that occur during call processing.

It now follows from the ***stochastic*** *decomposition* property of the M/G/1 queue with server vacations [1] that

$$\begin{aligned} \tilde{t}_C(s) &= \tilde{t}_{CI}(s) \frac{1 - \tilde{v}_I(s)}{sEV_I} \\ &= \frac{s(1 - \rho_{CI})\tilde{x}_I(s)}{s + \lambda_C - \lambda_C \tilde{x}_I(s)} \frac{1 - \tilde{v}_I(s)}{sEV_I} \end{aligned} \tag{12}$$

where $\frac{1-\tilde{v}_I(s)}{sEV_I}$ is the L.S.T. of the c.d.f. of the residual lifetime random variable of V_I [9].

3.4.3 Call busy periods

For calculation of call busy period distribution, we consider a vacation in which at least one arrival has occurred.

Note that the presence of vacations in the model causes the elongation of call busy periods because an arrival to an empty system does not immediately start service, but waits for the completion of the vacation time Other arrivals may occur during this residual vacation and these will contribute to the following busy period. Let G_{CV} denote the random variable for the busy period due to calls arriving during such a vacation and $\tilde{g}_{CV}(s)$ be its L.S.T.

Let $\tilde{g}_C(s)$ denote the L.S.T. of the call busy period distribution in the model without vacations. We have [4],

$$\tilde{g}_C(s) = \tilde{x}_I(s + \lambda_C - \lambda_C \tilde{g}_C(s)) \tag{13}$$

Conditioning on V_I and then unconditioning we get

$$\tilde{g}_{CI}(s) = \frac{\tilde{v}_I(\lambda_C - \lambda_C \tilde{g}_C(s)) - \tilde{v}_I(\lambda_C)}{1 - \tilde{v}_I(\lambda_C)} \tag{14}$$

where $\tilde{v}_I(s)$ is as in the previous section.

3.4.4 Cycle time for F fillwork segments (vacations)

Let U denote the random variable denoting the time until one cycle of F fillwork segments is completed and a fresh cycle starts and let $\tilde{u}(s)$ denote **its L.S.T.**

A vacation is followed by a call busy period if arrivals take place during the vacation. Otherwise, the server **goes** on another vacation. Owing to the assumption of Poisson arrivals, we have a renewal process with a renewal epoch defined at the start of a vacation.

Let Z denote the random variable for the interrenewai time and $\tilde{z}(s)$ its L.S.T. Conditioning on the vacation length random variable V_I,and recalling $\tilde{g}_C(s)$ from Equation **13,** we have

$$\tilde{z}(s) = \tilde{v}_I(s + \lambda_C(1 - \tilde{g}_C(s))) \tag{15}$$

Finally, $\tilde{u}(s)$ **is** obtained from $\tilde{u}(s) = \tilde{z}(s)^F$

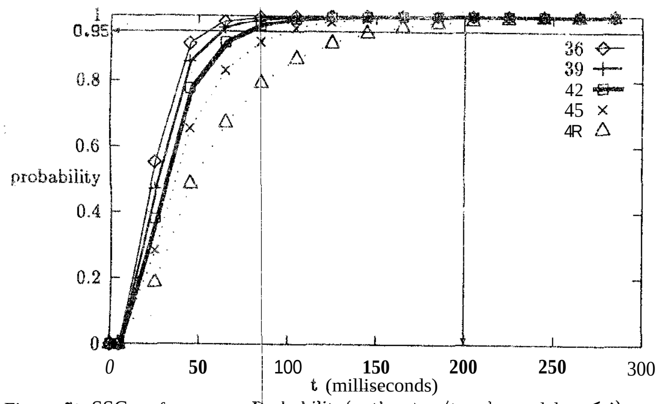

Figure 5: SSC performance: Probability(path setup/teardown delay $< t$) vs. t. $\alpha_1 = 0.95, t_1 = 200$ms.

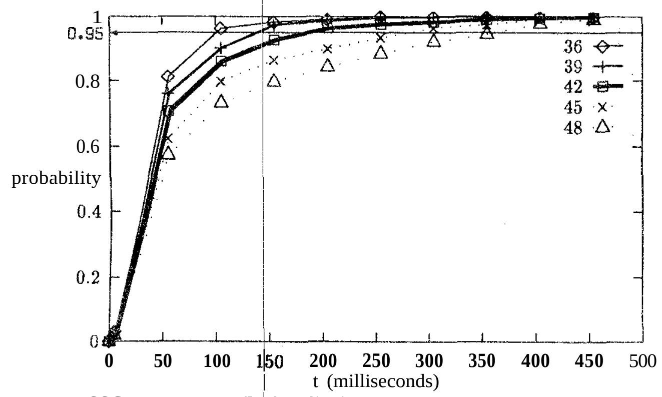

Figure 6: SSC performance: Probability(call processing busy period $< t$) vs. t. $\alpha_2 = 0.95, t_2 = 5$ rec.

3.5 Occupancy measurements in the SSC

We note here briefly our resolution of an anomaly that was observed by C-DoT engineers when they measured SSC processor occupancy. Whereas processor 'occupancy should increase linearly with λ, as is clear from Equation 1, the measurements yielded a nonlinear, convex, increasing curve.

Upon investigation, it turned out that the measurement process was measuring the fraction of time the SSC queue in nonempty. This is what we have denoted as ρ_{CI} (Equation 11), and is clearly nonlinear in λ. The reason for the nonlinearity is clear: as λ increases, so does λ_I, thus causing an increase in the effective mean service time of a call request (i.e., EX_I).

3.6 Numerical Results

The transforms obtained in the previous section were inverted using a numerical inversion technique for Laplace transforms [3]. The numerical data used are as follows: $\sigma = 0.735$, $b_{setup} = 10ms$, $b_{teardown} = 6ms$, $b_{int} = 0.6ms$, $b_{fill} = 10ms$, $t_1 = 200ms$, $t_2 = 5$ seconds, $t_3 = 10$ seconds, $F = 100$, $\alpha_1 = \alpha_2 = \alpha_3 = 0.95$.

The cumulative distribution functions for the various performance measures were obtained for a set of λ values corresponding to 70%, 75%, 80%, 85% and 90% occupancy of the SSC. These values of ρ correspond, respectively, to $\lambda = 36, 39, 42, 45, 48$ call attempts per second.

Figure 5 shows the probability distribution of the call setup/teardown delay in the SSC for the various values of BHCA mentioned above. It is clear that space switch path request delay requirements are easily met upto an attempt rate of 48 calls/sec, i.e., 172,800 BHCA, which corresponds to 90% processor occupancy.

Figure 6 shows the distribution of call processing busy period. Clearly this performance requirement does not constrain SSC capacity at all.

Figure 7 shows the cycle time for completing 100 fill work segments. This graph shows that at 45 calls/sec this requirement is met, but at 48 calls/sec it is not.

Owing to certain operational requirements, the possibility of adding other kinds of work to the SSC, and due to just plain caution, it is not considered appropriate to operate at above 70% occupancy. All performance requirements are, of course, met at this occupancy corresponding to a load of 129,600 BHCA

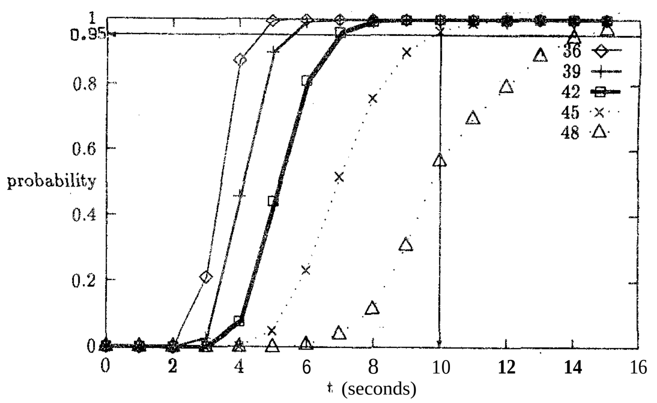

Figure 7: SSC performance Probability(completion time for 100 fill-work segments $< t$) vs t $\alpha_3 = 0.95, t_3 = 10$sec

4 Performance Analysis of the Administrative Processor

From a modelling perspective, the AP is very similar to the SSC.

4.1 The AP and its activities

Recall from Section 2 that the AP is connected to the CMS through a 128 kbps link and is the interface between the switch and the system administration consoles. Call processing work at the AP involves routing and allocation/deallocation of trunk resources. In addition, the AP acts as a conduit for flow of billing and traffic information from the BMs to the secondary storage at the Input Output Processor(IOP). The AP communicates with the other module processors through messages that are switched by the CMSs. The AP involves itself in processing the hardware link interrupts generated by these message transfers. Like every other processor, the AP participates in basic maintenance activities which include the reception of sanity punches from its copy (the AP is duplexed), the SSC, the IOPs and the Alarm and Display Unit (ADP). The AP in addition performs audits during idle time.

4.2 AP occupancy

We define the activities of call processing, sanity punch processing and interrupt processing its constituting the nondeferrable work at the AP. Deferrable work comprises the collection and delivery of billing/traffic records to the IOP, and also audits.

AP occupancy is the fraction of time the AP spends on nondeferrable work. The methodology for expressing the occupancy as a function of the call attempt rate is similiar to that of the SSC.

Since the discussion in this section is entirely about the AP, we can adopt the same notation as in Section 3, without causing any confusion

Let

b_{alloc} = service time for routing and trunk allocation

$b_{dealloc}$ = service time for trunk deallocation

Each intermodule call generates 3 messages at the AP: a route-request from the originating BM to the AP, a path setup request from the AP to the SSC, and finally, on call completion, a resource deallocation request from the originating BM to the AP. It follows, as in the case of the SSC, that

$$\rho = \rho_{SMP} + \sigma\lambda(b_{alloc} + b_{dealloc}) + \lambda_I b_{int} \tag{16}$$

and

$$\lambda_I = 12\sigma\lambda + \lambda_{I_{dump}} + \lambda_{I_0}$$

where $\lambda_{I_{dump}}$ is the interrupt rate due to billing and subscriber monitoring dumps from the BMs to the IOP via the AP, and has the form

$$\lambda_{I_{dump}} = 0.3\sigma\lambda + 0.16\lambda$$

The coefficients in this equation are obtained from analysis of the AP interrupts due to dumps, and $\lambda_{I_0} = 8.5$ per second for a 32 BM configuration[7].

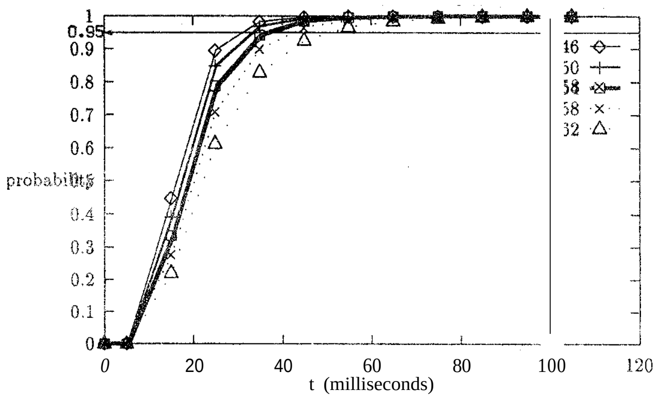

Figure 8: AP performance: Probability(call sojourn time at A? $< t$) vs. t. $\alpha_1 = 0.95$. $t_1 = 100$ ms.

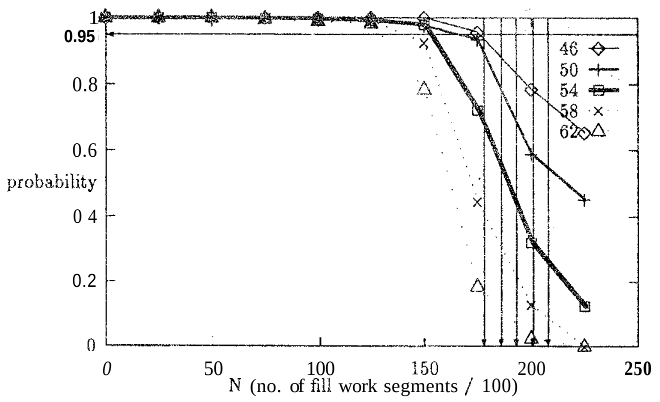

Figure 9 : AP performance: Probability(number of till work segments done in 60 min $> 100N$ vs. N. $\alpha_2 = 0.95$. (The vertical arrows point to the fill work requirement at each BHCA. This requirement increases with BHCA.)

4.3 Queuing model for the AP

For the purposes of modelling, work at the AP is considered to be at three priority levels:-

1. Interrupts, caused by message transmission and reception over the iinks, are served at hardware priority level. These are preemptive in nature.
2. Call processing work that involves routing arid trunk allocation/deallocation is served at the highest software priority level
3. Billing, traffic dumps and audits are served at the lowest priority.

Thus we again have a three priority mode: as with the SSC. Again, we treat the low priority work as fill work that is done in 10 ms. segments. The analysis is identical to that in Section 4.

The performance measures identified are call sojourn times and the probability that F fill work segments are completed within a given duration, which translate into the following performance requirements:-

$$P(call\ \text{sojourn}\ time\ \text{at}\ the\ AP\ < t_1) > \alpha_1 \quad (17)$$

$$P(cycle\ lime\ for\ F\ fill\ work\ \text{segments}\ < t_2) > \alpha_2 \quad (18)$$

4.4 Numerical results

The numerical data used for obtaining occupancy arid performance measures are as follows: $\sigma = 0.735$, $b_{alloc} = 7ms$, $b_{dealloc} = 3ms$, $b_{int} = 0.6ms$, $b_{fill} = 10ms$, $t_1 = 100ms$, $t_2 = 3600sec$, $a: = \alpha_2 = 0.95$

In Figure 8 the c.d.f. of call sojourn time in the AP is plotted for $\lambda = 46$, 50, 54, 58, 62 calls/sec, which correspond, respectively, to $\rho = 0.60, 0.65, 0.70$, 8.75, 0.80. It is seen from the graph that sojourn time requirements are met even upto $\rho = 0.80$.

In Figure 9 we plot the probability of performing $100N$ fill work segments in 3600 seconds versus N, for $\lambda = 46, 50, 54, 58, 62$ calls/sec. The amount of fill work to be performed in one hour depends on A, since the billing and traffic dumps are traffic. relat :d. Consequently, the value of F in the performance requirement (Equation 18) depends on A. For each λ the corresponding value of F is shown as an abscissa in Figure 9, the values increasing with BHCA. It is clear that only for $\lambda = 46$ ($\rho = 0.6$) is the perforrnance requirement met.

It follows that the capacity of the AP is 165,600 BHA at 60% processor occupancy.

5 System Overload Control

In this section, we propose and partialiy develop a methodology for designing an overload control strategy for the entire switch. We only have space here to provide a summary of our work; for details see [7] and [6].

Recall that the capacity of each processor module corresponds to a maximum occupancy. If the load offered causes an occupancy exceeding this, an overload control mechanism must respond quickly and attempt to shed the excess load. The objective of such a load shedding strategy will be to maximise carried load, while giving satisfactory performance to all the carried load.

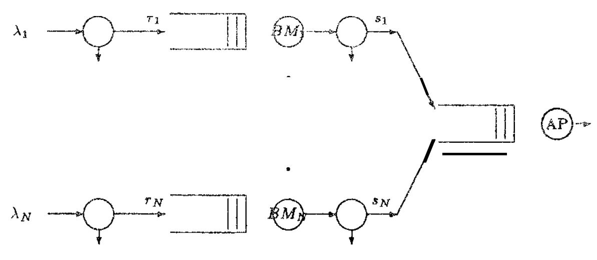

Figure 10 Queuing model for the overload control problem. The "wive"-like symbols denote load shedding points

5.1 Formulation and solution of the overload control problem

In the work that we report here, we have taken the first step towards devising an integrated control strategy for protecting all the processor based modules. For simplicity, we begin by considering only the BPs arid the AP. This motivates the model in Figure 10 with which all the subsequent discussions are concerned.

There are N BMs. In Figure 10, λ_i denotes the rate of calls entering BM_i. Prior to each BM, there is a $pre-dial$ control that controls the rate into BM_i to $r_i (\leq$ A,). Between each BM and the AP there is possibly a $post-dial$ control that further limits the rate r_i from BM_i to $s_i (\leq r_i)$. Thus the total input rate into the AP is $\sum_{i=1}^{N} s_i$.

The mean service times per call at the BP and AP are denoted respectively by b and a. We are given $0 < \rho^*_{BP} < 1$ and $0 < \rho^*_{AP} < 1$, such that the occupancies of each. BP and AP must be maintained less than ρ^*_{BP} and ρ^*_{AP} respectively. Thus, all the offered load can be handled by rach of the processors, if $\lambda_i \leq \rho^*_{BP}/b$, for all i, and $\sum_{i=1}^{N} \lambda_i \leq \rho^*_{AP}/a$. It follows that the maximum carried load $\bar{\gamma}$ for given $(\lambda_1, \ldots, \lambda_N)$ is given by $\bar{\gamma}(\lambda_1, \ldots, \lambda_N) = \min((\sum_{i=1}^{N} \min(\lambda_i, \rho^*_{BP}/b)), \rho^*_{AP}/a)$.

The problem is to decide at which points control should be exercised, and for a given set of values $(\lambda_1, \ldots, \lambda_N)$ what the values of $(r_1, \ldots, r_N)$ and $(s_1, \ldots, s_N)$ should be. We will impose the constraint that the throughput of the system should be as large as possible, i.e., $\bar{\gamma}(\lambda_1, \ldots, \lambda_N)$. Clearly there are many ways of shedding load to achieve this maximum throughput.

In a post-dial scheme, such as that of AT&T's 5ESS, though the contrcl can react rapidly to an overload, it is clear that under sustained overload, serving calls at She BMs oniy to reject them later is wasteful of the BM capacity. It would be good to have a mechanism that rejects calls right at their entry point to the BM, i.e., a pre-dial control. In such a case, we can have reduced dial-tone delays. In fact, our simulation results have shown that even the transient performance of tho post dial control is not superior to the controls we devise.

At variance with the adhoc techniques used in earlier and existing switches, we have given a novel treatment for overload control by formulating it as a problem of finding the amount of load to be accepted at each BM. so that average end-to-end call setup delay is minimised, while keeping: the throughput as large as possible (i.e., at $\bar{\gamma}(\lambda_1, \ldots, \lambda_N)$). We shall assume that all the call rejection is done before the BPs.

Let N, 6, and a be as defined earlier in this section. Let

x_i = call arrival rate to BP_i.

r_i = accepted call rate at BP_i.

We will first assume that x_i is a load that can be safely handled by BM_i (i.e.,$x_i = \min(\lambda_i, \rho^*_{BP}/b)$). Assume the presence of a pre-dial control for all the BMs that lets in loads that are within their capacities. Thus essentially we will first show how to further limit the load to protect the AP.

Now it can be shown ([7], [6]) that if the mean delay curves of the BPs are convex, and nondecreasing, then in order to minimise mean call setup delay while keeping throughput as large as possible (i.e., $\min(\sum_{i=1}^N x_i, \rho^*_{AP}/a)$), the optimal pre-dial control has the following form

$$r^*_i = \min(x_i, r^*), \qquad 1 \leq i \leq N$$

where r^* is the solution of

$$\sum_{i=1}^{N} \min(x_i, r^*) = \min(\sum_{i=1}^{N} x_i, \rho^*_{AP}/a) \tag{19}$$

Observe that the form of the optimal solution naturally lends itself to a threshold control implementation. The ideal throughput of such a controller is given by

$$\begin{aligned} r_i &= x_i \quad \text{if } x_i < r^* \\ r_i &= r^* \quad \text{if } x_i \geq r^* \end{aligned}$$

A threshold controller is realised as follows [5], and does not require knowledge of the x_i. Consider the presence of a token pool at the BM. Each call that enters the BM is accepted for service at the BM, only if the pool is not empty. The call takes away a token. Arriving calls that find the pool empty, are rejected. When the threshold is r^*, a token is added to the pool every $1/r^*$ units of time. The token pool can hold a maximum of M tokens. The ideal throughput is achieved with $M = \infty$. Thus to keep the throughput as close to ideal as possible, a large stack is desirable. This, however, leaves open the possibility of too large a number of calls being dispatched to the BM in a short period of time, i.e., the possibility of burstiness leading to larger queueing delays at the BP queue. Thus the value of the maximum token count is a compromise between throughput and delay.

5.2 On-line determination of the optimal control level

We need a control algorithm that obtains r from on-line measurements. The value of r should closely track the optimal control r^* as the offered load varies.

For fixed $(\lambda_1, \ldots, \lambda_N)$, i.e., fixed $(x_1, \ldots, x_N)$, define $f(r) = \sum_{i=1}^N \min(x_i, r)$. Observe that $f(r)$ is the rate of calls reaching the AP when the threshold control at each of the BMs is set to r. The problem of obtaining r^* (Equation 19) can now be formulated as that of obtaining the projection of the root of $f(r) - \rho^*_{AP}/a = 0$ into the interval $[\rho^*_{AP}/(aN), \rho^*_{BP}/b]$ [7]. Note that "noisy" measurements of $f(r)$ can be obtained by measuring the occupancy of the AP over successive intervals. The control level is adjusted at the ends of these measurement periods.

Thus the problem becomes one of finding a projection of the zero of a certain function $f(r)$ whose noisy values can be obtained for each value of its argument r. The same problem was obtained in the context of post-dial control of the CP in the 5ESS switch, and well known *stochastic approximation* techniques have been appiied to it. by the first author In [5]. If y_k is the measurement of $f(r_k)$, (i.e., at the kth step of the iteration), the algorithm yields r_{k+1} as follows

$$r^{-1}_{k+1} = \theta_{k+1} = \left[\theta_k + \frac{r_k}{\sum_{i=k-W+1}^{k} r_i^2}(y_{k+1} - 1)\right]_{b/\rho^*_{BP}}^{(aN)/\rho^*_{AP}} \tag{20}$$

where $[\cdot]_{\underline{\theta}}^{\bar{\theta}}$ is the projection of the quantity in $[\cdot]$ into the interval $[\underline{\theta}, \bar{\theta}]$. The sequence $\{\theta_k\}$ is an intermediate quantity whose interpretation can be found in [5].

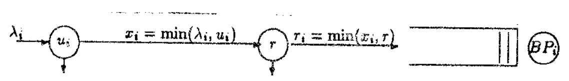

Figure 11 Schematic of independent BP and AP controls

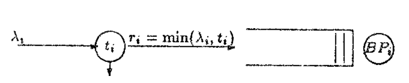

Figure 12 Schematic of integrated Mi. and AP controls

5.3 Integrated overload control for the BPs and the AP

We have thus far assumed that the offered loads to each of the BMs are within their capacities. Now suppose that the BP overload controls are also implemented as threshold controls (as shown in Figure 11). Let

u_i = the threshold at BP_i for protecting BP_i

r = the threshold that is applied at each BP to protect the AP.

λ_i = rate of calls offered to BP_i.

Letting $t_i = \min(u_i, r)$, we see that both the controls can be integrated intc a single threshold control, as shown in Figure 12. This suggests that both the AP and the BM in question work on the control algorithm independently and determine the values r and u_i. The BM then applies a rate min(u_i, r) as the control.

5.4 Simulation experiments and results

A series of simulation experiments were done to study the proposed overload control.

Each of the N BMs is modelled as a single server queue. The BMs are all identical, with equal service speeds. Calls arrive at the BM in a Poisson process. We model a call as follows: Each call generates a random number of stimuli. A stimulus requires an exponentially distributed service time of b_{stim} millisecs. Each stimulus, after its service time of b_{stim} millisecs has been expended, goes into a wait state with a probability p where $1/(1-p)$ = average number of stimuli per call. The wait state is represented by a delay node with a delay distribution that is exponential with mean t ms. Thus the mean service time of a call at the BM is $b = \frac{b_{stim}}{1-p}$.

The Administrative Processor (AP) is modelled as a single server $\cdot/M/1$ queue. Calls departing from the queues of the various BMs enter this queue. Each call requires a millisecs at the AP. We assume that each RM is offered a load all of which requires service at the AP also. Note that this assumption is not strictly true, since a BM is subjected to load due to both originations and terminations. Only originations eventually require service at the AP. Terminating calls at a BM require service oniy at the BM. Further work is needed to incorporate termination traffic into our model.

The numerical data used are as follows: A' = *30*, $\rho^*_{AP} = 0.7$, $\rho^*_{BP} = 0.7$, Maximum value for the threshold counter $M = 30$, $a = 10ms$, $b_{stim} = 20ms$, $t = 500ms$, $p = 0.9$, Measurement and control update interval = 5 seconds.

We report here the results of an experiment that was done to study the effect of the location of the control points, (pre-dial or post-dial or both) and to study their relative performance with respect to the steady state and transient bchaviour. The loads on tire BMs were changed after every 3 minutes to study the adaptiveness of the control strategy shown in Equation 20.

For the first 3 minutes, the loadson the BMs were $\lambda_1 \ldots \lambda_{20} = 3.5, \lambda_{21} \ldots \lambda_{30} = 1$. The correct level of control is 3.0 For the next 3 minutes, the loads were $\lambda_1 \ldots \lambda_{30} = 3.0$. The correct level of control is *2.33*. For the last 3 minutes of the simulation, the loads were $\lambda_1 \ldots \lambda_{30} = 2.0$. The correct level of control is 4.5.

control scheme	mean dial-tone delay (ms.)	mean delay in AP (ms.)
I	45.6	29.0
II	40.4	22.7
III	39.8	21.6

Table 1: Mean dial-tone delay and mean delay in AP for the three overload control schemes.

Three alternatives were studied:

- *Pre-dial rate control for the BM and a post dial rate control for the AP:* This simulation was done to evaluate the performance of the post-dial scheme used in the 5ESS. Let this scheme be called Scheme I.
- *Pre-dial rate control for the BM and a pre dial plus a post dial rate control for the AP:* This control strategy was tested to see the advantage in having both a post-dial mechanism that may have the advantage of a better transient behaviour and a pre-dial mechanism that has better steady state behaviour. Under sustained oveload, the post-dial control will not have any effect since the pre-dial control would already have limited the loads to satisfy the AP requirements. We shall call this Scheme II.
- *Pre-dial rate control for the BM as well as the AP:* This is Scheme III.

We obtained plots of the control level and measured occupancy versus time for the 3 minute duration of each of the 3 experiments. Owing to lack of space we cannot show these here (see [6]). All three schemes track the correct control level, but there are differences in the convergence times, Scheme I being the slowest. We also obtained the mean dial-tone delay (this is taken to be the mean time for serving the first stimulus in the call model described above), and the mean delay in the AP for each experiment. These are shown in Table 1.

It is seen from Table 1 that Scheme I gives the worst performance in terms of average delays at the BP and the AP. Scheme 11 gives a performance close to that of Scheme III, but is slightly inferior. We now compare the schemes I and III. The strategy used in the 5ESS namely a pre-dial control and an independent post-dial control for the AP gives BP delays higher than that of the scheme that uses only a single pre-dial control for the BPs as well as the AP. This is of course expected since in Scheme I the BPs are unnecessarily handling load that is later rejected by the post-dial control. Scheme I, however, also gives a slightly higher delay at the AP. This is quite unexpected because a post dial scheme should react to a transient condition faster. The reason for the poor performance can be attributed to the finite convergence time of the algorithm in responding to the change in load. If we could have a hypothetical situation in which the AP could immediately detect an overload, then Scheme I would indeed give lower delays at the AP.

Work remains to be done to use this methodology have to develop a comprehensive overload control strategy, that protects all the processor modules, and accounts for all the types of traffic that modules are subjected to. In particular, we have not acconnted for termination traffic in the BPs; further our model does not incorporate the SSC. Our methodology establishes a conceptual framework, and it should not be difficult to incorporate these other factors.

Acknowledgement

We are grateful to several C-DoT engineers for answering patiently our many questions about the hardware and software architecture of the switch, and for providing us with various measurements. In particular, we wish to thank Mr. Venkatrajendran, Divisional Manager, Software, and the members of the Performance Engineering Group: Mr. Srinivasachar, Ms. Naseein Meriam, and Mr. Venugopal.

References

[1] S. W. Fuhrmann and R.B. Cooper,"Stochastic decomposition in an M/G/1 queue with generalised server vacations", Operations Research, Vol.33, 1385.

[2] B. Hareesh Kumar, "Performance analysis and optimisation of the C-DoT digital switching system", Master of Engineering Project Report, Dept. of Electrical Communication Engg., Indian Institute of Science, Bangalore, January 1990.

[3] D.L. Jagerman,"An inversion technique for the Laplace Transform," Bell System Technical Journal, Vol. 61, No. 8, pp 1995-2002, Oct. 1982.

[4] L. Kleinrock, *Queueing Systems*, Vol. I and II, John Wiley.

[5] Anurag Kumar,"Adaptive overload control of the central processor in a distributedgystem with a star topology," *IEEE Transactions* on *Computers*, October 1989.

[6] Anurag Kumar, B. Hareesh Kumar, and B. Madhavi, "Modelling and performance analysis of a digital switching system - case study of the C-DoT DSS", to appear in *Journal of the Institution of Electronics and Telecommunications Engineers*, 1991-92.

[7] B. Madhavi, "Performance analysis of the SSC and the AP and design of the system level overload controls in the C-DoT Digital Switching System," Master of Engineering project report, Dept of Electrical Communication Engg., Indian Institute of Science, Bangalore, January 1991.

[8] A. Tanenbaum, *Compufer Networks*, II edition, Prentice Hall, 1988

[9] R.W. Wolff,*Stochastic Modelling and the Theory of Queues*, Prentice Hall, 1989.