A STRICT PRIORITY QUEUING SYSTEM WITH OVERLOAD CONTROL

Martin Eisenberg
Bell Telephone Laboratories

Abstract
The processor of an electronic switching system has many tasks to perform in real time. A recent switching system has been designed in which the scheduling scheme for the tasks is strict (i.e., non-preemptive) priority. This basic scheduling scheme must be modified at high load by an "overload control" mechanism that ensures an adequate execution rate of the lowest-priority jobs. This paper describes two models that have been used to investigate the load-dependent behavior of the system: a non-preemptive priority queuing model to analyze the processor, and a Markov chain model to analyze the overload control mechanism.

1. Introduction
Every electronic switching system must have an algorithm for deciding which, of the many jobs that may be ready for execution, should next be given control of the processor. The scheduling algorithm must take into account the fact that different job types have widely different tolerances for delay. For example, digit reception and off-hook recognition require quick response lest calls be misrouted or lost; on the other hand, jobs such as routine audits and maintenance may be delayed without ill effect.

The scheduling algorithm must be prepared to handle the case of overload, when the rate of arriving calls exceeds the capacity of the processor to handle all calls. During overload, the objective of the scheduling algorithm differs from its objective during normal load: At ordinary load its objective is to minimize call-processing delays, especially dial-tone delay (the time between "off-hook" and the delivery of dial tone to the caller); during overload, the objective is to maximize the number of successfully handled calls. To ensure that calls that have already entered the system can be successfully handled during overload, it is necessary either to delay, or prevent altogether, the acceptance of (i.e., the delivery of dial tone to) new calls.

While the term "scheduling system" should perhaps refer to the method of scheduling jobs under all load conditions, in practice the term normally refers only to that system that is used at ordinary load. We use the term "overload control" to refer to the change of the basic scheduling system during overload.

In this paper we describe two models that have been used to study the behavior of the processor of an electronic switching system using a non-preemptive priority discipline as the basic scheduling scheme, together with a special overload control mechanism that was designed specifically for use with the strict priority basic schedule.

2. The Scheduling System and Overload Control
The basic scheduling scheme we consider is a non-preemptive priority discipline with periodic "interrupts," during which certain time-critical jobs are executed. The processor time used for these time-critical jobs is referred to as the foreground, while the main work of the system is said to be executed in background.

Each background job is assigned a priority depending on its job type. Call-processing jobs, for example, are assigned a relatively high priority, while routine audits and maintenance jobs are assigned a low priority. Long jobs are divided into short segments so that jobs that arrive when a lower-priority job has control can be executed without having to wait for the lower-priority job to be completed. When there is no other work to be done, processor control is given to jobs called fill work. Fill-work jobs are useful, but non-essential, and are assigned the lowest priority.

At low to moderate load, this basic non-preemptive priority scheduling system works well, since it minimizes the delays experienced by callers. For satisfactory operation, however, the basic scheduling system must be modified at high load using an overload control mechanism. Otherwise, if the call arrival rate became high enough, processor time would be completely taken up with high-priority call-processing work, and the processor would be prevented from executing low-priority, but essential, jobs. At heavy
load, it becomes necessary to delay some new calls before they enter the system, or to block some calls altogether, so that the work at the lowest priorities (other than fill work) can be completed. The overload-control mechanism used to accomplish this is called the throttle.

The operation of the throttle is as follows: Two throttle numbers, \( l \) and \( c \), are maintained by the system. When a new call arrives, the status of these throttle numbers is checked. If either or both of the throttle numbers is positive (we discuss shortly how they become positive), the newly arriving call is immediately admitted into the system. Simultaneously, a throttle number \( c \) is decremented by one. If both \( l \) and \( c \) are equal to zero when the call arrives, the call is placed in the Input Buffer, where it waits to be admitted. From this description it is clear that the throttle numbers represent, at any given time, the number of newly arriving calls that will be accepted without delay.

At regular intervals (every \( v \) milliseconds) the system executes the "throttle process", which first adjusts the throttle numbers and then decides if any calls waiting in the Input Buffer should be admitted into the system. If either \( l \) or \( c \) becomes positive by the adjustment, as many calls as allowed (up to \( l + c \) ) are removed from the Input Buffer and admitted into the system. The value of \( l \) or \( c \) is reduced by one for each call that is admitted (reducing \( c \) before \( l \)).

Throttle number \( c \) is intended to be the primary overload control mechanism. The way that this variable is adjusted is as follows: During each execution of the throttle process, a test job is generated and inserted into the background job queue at a low priority level—a level just above fill work. The test job performs no function other than recording whether it has been executed, i.e., whether the processor was able to empty the system of all higher-priority jobs. At each execution of the throttle process, a check is made to see whether the test job was run since the last execution of the throttle process. If the test job was executed at least once during the previous \( v \) milliseconds (a successful test) \( c \) is incremented by one. If the test job was not executed during the previous \( v \) milliseconds (an unsuccessful test) \( c \) is not adjusted. Every \( l \) executions of the throttle process, this test is not performed; instead, throttle number \( c \) is simply reset to zero.

From this description it should be clear that throttle number \( c \) acts in a "self-correcting" manner: At high load, when there is insufficient real time to allow the processor to reach the lowest priority jobs, \( c \) is not incremented frequently and new calls tend to be delayed, thus allowing the processor more time to execute its non-call-processing work; at low load, when the processor reaches the lower priorities frequently, throttle number \( c \) will be incremented rapidly, and new calls will tend to be admitted without delay. Note that the periodic resetting of \( c \) to zero is done to prevent \( c \) from growing so large during a lightly loaded period that insufficient overload protection is available during a later heavily loaded period.

In addition to the \( c \) method of control, the throttle allows a certain number of calls into the system independently of the real-time condition of the processor. These calls enter via the \( l \) throttle number. Every \( l \) executions of the throttle process (every \( l v \) milliseconds), the value of \( l \) is reset to \( l_0 \). It can be seen that, at high load, throttle number \( l \) acts to admit a steady rate of calls into the system (\( l_0 \) calls every \( l v \) milliseconds).

The throttle overload-control mechanism is specifically designed for a processor using the strict priority service discipline. Older systems, in which the service discipline is cyclic priority, require a different method of overload control (cf Farber [1]).

3. Behavior of the System

Depending on the characteristics of the system and on the parameters associated with the throttle \((l, l_0, v, \lambda)\), the throttle has a maximum rate at which it will admit calls. We call this maximum rate the throughput of the throttle, and denote it by \( \lambda_T \). If the call arrival rate is large enough to keep the Input Buffer full, then the throttle will accept calls at its throughput rate.

![Figure 1. Admitted Load Versus Offered Load.](image)

Figure 1 shows the rate at which new calls are admitted into the system by the throttle as a function of the rate at which calls arrive. When the call arrival rate is less than \( \lambda_T \), all arriving calls are (ultimately) admitted. When the arrival rate is greater than \( \lambda_T \), some calls are prevented from entering the system altogether. The point denoted \( \lambda_0 \) in the figure is the "100%-occupancy" arrival rate, i.e., the call rate that, if switched, would allow the processor zero time for fill work. If the throttle is to operate properly, its parameters must be set so that \( \lambda_T < \lambda_0 \). Clearly, at a call arrival rate near the throughput rate, dial-tone delays will be large, and will not satisfy normal criteria.
There is some maximum call arrival rate for which dial-tone delays are acceptable; this rate is called capacity, and is denoted by $\lambda_C$ in the figure.

4. The Queuing Model

In this section we describe a priority queuing model that has been used to investigate the behavior of the strict-priority scheduling system. The notation we use is as follows: The number of priorities is $P$, where priority 1 is the highest priority and Priority $P$ is the lowest. The job arrivals at each priority are assumed to be Poisson; we denote the arrival rate to priority $p$ by $\lambda_p, p=1,\ldots,P$. The service-time distribution function at priority $p$ is denoted by $F_{sp}(t)$, and the Laplace-Steiltjes transform of $F_{sp}(t)$ is denoted by $S_p(s)$. The mean and second moment of $F_{sp}(t)$ are $1/\mu_p$ and $E(S_p^2)$, respectively.

It is a classical queuing theory result[2] that the Laplace-Steiltjes transform of $F_{wp}(t)$, the waiting-time distribution function at priority $p$, is

$$W_p(s) = \frac{(1-p)[s+\lambda_{H_p}-\lambda_{H_p}B_{H_p}(s)]}{s-\lambda_p+\lambda_pS_p(s+\lambda_{H_p}-\lambda_{H_p}B_{H_p}(s))}$$ (1)

where

$$\rho = \sum_{p=1}^{P} \frac{\lambda_p}{\mu_p}$$ (2)

is the total utilization of the system,

$$\lambda_{H_p} = \sum_{j=1}^{P-1} \lambda_j$$ (3)

is the arrival rate of jobs of priority higher than $p$,

$$\lambda_{L_p} = \sum_{j=p+1}^{P} \lambda_j$$ (4)

is the arrival rate of jobs of priority lower than $p$,

$$S_{L_p}(s) = \sum_{j=p+1}^{P} \frac{\lambda_j}{\lambda_{L_p}}S_j(s)$$ (5)

is the Laplace-Steiltjes transform of the effective service-time distribution function for jobs of priority lower than $p$, and $B_{H_p}(s)$ is the Laplace-Steiltjes transform of the effective busy-period distribution function for jobs of priority higher than $p$. The busy-period transform satisfies the implicit relationship

$$B_{H_p}(s) = S_{H_p}(s+\lambda_{H_p}-\lambda_{H_p}B_{H_p}(s))$$ (6)

where

$$S_{H_p}(s) = \sum_{j=1}^{p-1} \frac{\lambda_j}{\lambda_{H_p}}S_j(s)$$ (7)

is the Laplace-Steiltjes transform of the effective service-time distribution function for all jobs of priority higher than $p$.

By differentiating Eq.(1) and evaluating the result at $s = 0$ we obtain another classical result[2]: The mean delay for priority $p$ is

$$W_p = \frac{\sum_{j=1}^{P} \lambda_j E(S_j^2)}{(1-\rho_p)(1-\rho_{p-1})}$$ (8)

where

$$\rho_p = \sum_{j=1}^{p} \frac{\lambda_j}{\mu_j}$$ (9)

is the utilization for priority $p$ and higher.

Our queuing model expands on these basic results in several ways. First, the model accounts for the fact that in an actual switching system the processor is never idle, but always executes a "fill" job whenever it has nothing else to do. This is accounted for in the model by adjusting the arrival rate of the lowest priority job, $\lambda_p$, to be such as to cause the total utilization, $\rho$, to be equal to one. Thus, the Laplace-Steiltjes transform of the waiting-time distribution function at priority $p$ becomes

$$W_p(s) = \frac{\lambda_{L_p}[1-S_{L_p}(s+\lambda_{H_p}-\lambda_{H_p}B_{H_p}(s))]}{s-\lambda_p+\lambda_pS_p(s+\lambda_{H_p}-\lambda_{H_p}B_{H_p}(s))}$$ (10)

The queuing model accounts for the effect of foreground interrupts in the following way: Assume that the time between the end of one interrupt and the beginning of the next interrupt is exponentially distributed with mean $1/\nu$, and assume that the duration of the foreground work after an interrupt is independently distributed with distribution function $F_D(t)$. Then, the "effective" duration of a service time in priority $p$, i.e., the length of a service from beginning to end, including foreground interrupts, denoted by $F_{Sp}(t)$, has the distribution function
The waiting time is equal to the service time as if the job were to arrive separately in a Poisson manner.

The queuing model calculates the distribution of the waiting time at each priority by numerically inverting the Laplace Transforms of Eq. (10), modified as described above. Figure 2 shows an example of the results of this model for a particular system and a particular load level. The delay distribution for priority 5 (the priority of the test job) gives the success probability of the test job as a function of the time between checks.

An example of the results of this model for a particular system and a particular load level. The delay distribution for priority 5 (the priority of the test job) gives the success probability of the test job as a function of the time between checks.

5. The Throttle Model

In this section we describe a Markov chain model that has been used to investigate the behavior of the throttle. (The model described in this section is a simplified version of the actual model.) The states of the Markov chain are specified by the values of the throttle numbers \( l \) and \( c \), the number of calls waiting in the Input Buffer, \( n \), and the phase of the throttle cycle, \( i \). (As described above, the throttle actions are performed in a cycle of \( I \) phases. During phases 1 through \( I - 1 \), the success of the test job is checked and \( c \) is adjusted accordingly. During phase \( I \), \( l \) is reset to \( l_0 \) and \( c \) is reset to zero.) The key assumption of the model is that the success probability of the test job in any interval is independent of its success in previous intervals. The method of solution is to write and solve the state equations for the quantities \( \pi_{ik} \), the probability that the state of the system is \( i, c, \) and \( n \), at the end of phase \( i \) of the throttle cycle. These probabilities are used to obtain the throttle delay.
distribution.

The state equations involve the quantity $p_k$, which is the probability that $k$ calls arrive between phases. We have

$$p_k = \frac{(\lambda v)^k}{k!} e^{-\lambda v} \quad (13)$$

where $v$ is the time between two successive throttle phases and $\lambda$ is the arrival rate of calls to the system.

The transition probabilities from phase $I-1$ to phase $I$ are as follows. (For simplicity, we omit the superscript $I-1$ before $\pi$ on the right-hand side of all the following equations.)

$$n > 0$$

$$l \pi_{00}^{I} = \sum_{k=1}^{n+I} \pi_{00}^{k} p_{n+I-k} + \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{00}^{\alpha} \right) p_{n+I-k} \quad (14)$$

$$n=0, \ 0 \leq l < l_0$$

$$l \pi_{l0}^{I} = \sum_{k=1}^{l-I} \pi_{l0}^{k} p_{l-I-k} + \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{l0}^{\alpha} \right) p_{l-I-k} \quad (15)$$

$$n=0, \ l=l_0$$

$$l \pi_{l0}^{I} = \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{l0}^{\alpha} \right) \sum_{j=0}^{k} p_j \quad (16)$$

Eq.(14) is explained as follows: There are two ways to leave $n > 0$ calls in the Input Buffer at the end of phase $I$. In the first way, there is at least one call in the Input Buffer at the end of phase $I-1$. We let this number be $k$. If $n+I_0-k$ calls arrive between the two phases, all the newly arriving calls will be placed in the Input Buffer. At the beginning of phase $I$, therefore, there will be exactly $n+I_0$ calls in the Input Buffer, of which $I_0$ will be accepted into the system because of the resetting of the $I$ throttle number.

In the second way, there are no calls in the Input Buffer at the end of phase $I-1$, and the sum of the throttle numbers is $k$, which can vary from $I$ to $I_0+I-1$. ($I-1$ is the maximum value that $c$ can obtain during phase $I-1$.) Now, if $n+I_0+k$ calls arrive between phases $I-1$ and $I$, the first $k$ of these will be accepted into the system immediately, and the next $I_0$ will have to wait until phase $I$ to be served, leaving $n$ in the Input Buffer.

For brevity, we omit the explanation of Eqs.(15) and (16).

The transition probabilities between all phases other than $I-1$ and $I$ are as follows:

$$n > 0$$

$$l \pi_{00}^{I} = \sum_{k=1}^{n+I} \pi_{00}^{k} p_{n+I-k} + \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{00}^{\alpha} \right) r_{n+I+k} \quad (17)$$

$$n=0, \ c=0$$

$$l \pi_{l0}^{I} = \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{l0}^{\alpha} \right) r_{n+I+k} \quad (18)$$

$$n=0, \ c>0$$

$$l \pi_{l0}^{I} = \sum_{k=0}^{I+I-1} \left( \sum_{\alpha+\beta=k} \pi_{l0}^{\alpha} \right) r_{n+I+k} \quad (19)$$

where

$r_0 = q_0$; $r_k = q_p + (1-q)pr_{k-1}$; 

and where $q$ is the success probability of the test job.

The number of calls in the Input Buffer at the end of phase $i$ will be $n > 0$ in two different ways, which account for the two terms of Eq.(17). In the first way, the number of calls in the Input Buffer at the end of phase $i-1$ is positive, equal to $k$. If the number of calls that arrive between phases is $n+I-1-k$ and the low-priority test is successful, then one call will be accepted into the system during phase $i$ and the number left in the Input Buffer will be $n$. Or, if the number of calls that arrive is $n-k$ and the test is unsuccessful, then no calls are accepted into the system and again the number left in the Input Buffer will be $n$.

The second way occurs when the number of calls in the Input Buffer is zero at the end of phase $i-1$. In this case, if the sum of the throttle numbers at the end of phase $i-1$ is equal to $k$, the first $k$ calls that arrive in the intervening interval will all be accepted into the system. If $n+I+k$ calls arrive and the test is successful, an additional call will be accepted during phase $i$, and $n$ calls will remain in the Input Buffer. If $n+k$ calls arrive and the test is unsuccessful, no additional calls will be accepted during phase $i$, and $n$ calls will remain the Input Buffer.
For brevity, we omit the explanation of Eqs. (18) and (19).

Equations (14) through (19) are the transition probabilities that relate the state probabilities at successive phases. The state probabilities are determined by iterative solution of the state equations: First, an initial guess is made for the values of the probabilities $1 \pi_k^m$ for all permissible values of $n, l, c$. Then, using the transition probabilities between phase 1 and phase 2, an initial estimate of $1 \pi_2^m$ is determined. This continues until an estimate of $1 \pi_2^m$ is determined. This continues until an estimate of $1 \pi_2^m$ is obtained. At this point, an updated estimate of the quantities $1 \pi_k^m$ is determined by using the transition probabilities between phase 1 and phase 2. The calculations are repeated until two successive estimates for the quantities $1 \pi_k^m$ differ by less than a specified amount.

Having obtained the state probabilities at each throttle phase, the next job is to determine the state probabilities at the instant of a call arrival. These quantities are needed to determine the delay distribution for a call. In particular, what is needed is the probability $w_k(t)$, which, if multiplied by $dt$, is the probability that the Input Buffer contains $n$ calls at a random call-arrival between phases $i$ and $i+1$, that $l = c = 0$ if $n = 0$, and that the time remaining until phase $i+1$ is between $t$ and $t+dt$. We have

$$
\omega_k(t) = \sum_{k=0}^{n} \pi_0^n P_{n-k}(t) + \sum_{k=1}^{n} \sum_{\alpha+\beta=k}^{l+1} P_{\alpha} P_{n+\beta}(t) \tag{21}
$$

where

$$
P_n(t) = \frac{1}{n!} \lambda^{n-t} e^{-\lambda (t-1), \quad 0 \leq t \leq \nu. \tag{22}}
$$

$P_n(t)dt$ is the probability for a call arriving between phases $i$ and $i+1$ that $n$ calls had arrived since phase $i$ and that the time remaining until phase $i+1$ is between $t$ and $t+dt$.

The first term of Eq. (21) represents the possibility that the Input Buffer had $k > 0$ calls at the end of phase $i$ and $n-k$ calls arrived between phase $i$ and the arrival of the observed call. The second term represents the possibility that the Input Buffer was empty at the end of phase $i$, the sum of the throttle numbers equaled $k$, and $n+k$ calls arrived before the observed call.

Now that the state probability at the instant of call arrival has been determined, it only remains to determine how long the call must wait to be accepted into the system. If a call arrives between phases $i$ and $i+1$, either it is accepted immediately or it must wait until at least phase $i+1$. If the call is delayed until phase $i+1$, it may then have an additional wait for acceptance, first to clear calls that are ahead of it, and then to reach a point where a throttle number becomes positive. The density function for the entire throttle delay, then, is

$$
\xi(t) = \frac{1}{I} \sum_{l=1}^{I} \left[ \sum_{k=0}^{I} \pi_k^0 \left( \sum_{\alpha+\beta=k}^{l+1} \sum_{j=0}^{l-i-k} P_j \right) w_0(t) \right] u_0(t) + \frac{1}{I} \sum_{l=1}^{I} \left[ \sum_{k=0}^{I} w_k(t) f_{n+1}^j(t) \right]
$$

where $P_j = \int_0^t P_j(t) dt$ is the probability that call arrives between the end of phase $i$ and the observed call, $u_0(t)$ is the unit impulse function, and $f_{n+1}^j(t)$ is the density function of the additional wait for acceptance of a call that is delayed until phase $i+1$, and that finds $n$ calls ahead of it in the Input Buffer. The first term of Eq. (23) represents the possibility that the observed call will see zero delay if it arrives after a phase that left the throttle numbers positive ($i+c=k$, $k \geq 1$) and if no more than $k-1$ other calls arrived before the observed call. The second term of Eq. (23) expresses the fact that the number of calls present at the arrival of the observed call is $n$, the time to phase $i+1$ has distribution $\omega_k(t)$, and the additional wait from phase $i+1$ to acceptance has the distribution $f_{n+1}^j(t)$. Note that later arrivals do not affect the delay of the observed call since the Input Buffer service discipline is first-come-first-served.

It remains only to obtain $f_{n+1}^j(t)$. This is a time function consisting of impulses at multiples of the interphase interval $\nu$. We obtain $f_{n+1}^j(t)$ by using the recursive relation

$$
f_{n+1}^j(t) = q^{n+1} \sum_{k=n}^{I} \left[ \left( 1-q \right)^k \right] u_0(t-(k+1)\nu) + \sum_{k=0}^{I-i-1} \left[ \left( 1-q \right)^k \right] u_0(t-(I-i)\nu) f_{n+1-i-k}^j(t) \tag{24}
$$

The first term of Eq. (24) is the probability distribution for the delay when the observed call finds fewer calls ahead of it than there are phases remaining to the end of the throttle cycle. In this case, the wait of the observed call will be $(k+1)\nu$ if (1) the $(k+1)^{th}$ test is successful, (2) exactly $n$ out of the preceding $k$ tests are successful. If $n$ exceeds $I-i-1$, this term is taken as zero; in this case, it is impossible to serve the call before the end of the throttle cycle.

The second term of Eq. (24) relates the delay at phase $i+1$ to the delay at the end of the cycle, if the call has to wait this long. In this case, the total delay
equals \((I-i)v\) (the time to the end of the cycle) plus the time to clear the calls that remain after \(k+l_0\) calls are served within the current cycle.

Equation (24) is solved recursively starting with \(n\) equal to zero, defining \(f_n^k(t)\) to be \(u_0(t)\) when \(n\) is less than zero.

Figures 3 and 4 illustrate results that were obtained with the throttle model. Figure 3 shows the average number of calls in the Input Buffer at the moment of throttle execution and at the instant of call arrival, plotted as a function of the phase of the throttle cycle. Specifically, the quantities plotted are

\[
\sum_{I,c,n} l \pi_{I,c,n}^n \quad \text{and} \quad \int_0^v \sum_{I,c,n} \omega_n^I(t) n \, dt.
\]

Figure 4 shows the average values of the throttle numbers plotted versus the phase of the throttle cycle, i.e.,

\[
\sum_{I,c,n} l \pi_{I,c,n}^n \quad \text{and} \quad \sum_{I,c,n} c \pi_{I,c,n}^n.
\]

6. Use of the Models

The queuing and throttle models have been used to set the throttle parameters and to determine system capacity. (The queuing model has also been of help in determining appropriate job priority assignments.) The way this was done is as follows: First, a value of \(\lambda_T\) was chosen, based on criteria for the minimum allowable execution rate of low-priority jobs. The queuing model was then employed using a call arrival rate of \(\lambda_T\). From this, we determined \(q(v)\), the probability that the test job is executed within the interphase interval \(v\), when the system switches calls at the throughput rate. The next step was to determine the proper values of the throttle parameters, \(I\), \(v\), and \(l_0\), such that the throttle would in fact have the correct throughput value \(\lambda_T\). This was done by satisfying the equation

\[
\lambda_T = \frac{l_0 + q(v)(I-1)}{Iv}.
\]  

Having determined the proper throttle parameter values, we used the queuing model and the throttle model at call arrival rates less than \(\lambda_T\) (the success probability \(q\) used in the throttle model being determined from the queuing model). The queuing and throttle delay distributions as a function of offered load were determined from these models. By convolving the appropriate delay distributions, the overall dial-tone delay distribution was determined as a function of call arrival rate. Finally, we determined the highest call arrival rate for which the dial-tone delay was satisfactory, and we thus were able to determine system capacity.

7. Conclusions

The models we have described have been used both to analyze the behavior of the switching system as well as to gain insight into how the system should be modified to improve this behavior. The results of our models were tested using simulations, and showed good agreement. Our results were used to optimize system performance, increase capacity, and reduce the sensitivity to variations in system characteristics.

References

Summary of Questions/Answers

Date: 09 June 1983
Session: 1.3
Paper: 2

Q.1 (D. Manfield)
Your analysis is done with two models, one for the priority queue, and a second for the throttle mechanism. Will not the throttle action have an effect on the arrival processes to your priority queue, and how much approximation is introduced into the analysis as a result?

A.1 (M. Eisenberg)
The existence of the throttle mechanism indeed has an effect on the priority queue. However, the jobs that enter the priority queue with the arrival of a new call do not all arrive simultaneously. In fact, the work associated with a particular call is spread out over a relatively long interval. (i.e. the period of dialing and circuit connection) Thus, the character of the arrival process to the priority queue is still very nearly Poisson, despite the effect of the throttle. This result has been tested using a simulation. Apparently very little error is introduced by our approximation.

Q.2 (Phuoc Tran-Gia)
How does the repeated attempts phenomenon influence the throttle mechanism?

A.2 (M. Eisenberg)
Calls that abandon and retry any number of times before they are removed from the Input Buffer cannot be distinguished by the system from calls which wait continuously to be served. Calls which abandon while still in the Input Buffer are removed from the system at the instant when they would otherwise be admitted. Subsequent reattempts appear as new calls.

Q.3 (Kalyan Basu)
To determine the proper values of throttle parameters Eq-25 is suggested. Can the author explain how I, V and LO are obtained.
A.3 (M. Eisenberg)

The design of the throttle has been modified as a result of this study. The parameter LO has been determined to be counter-productive and has been removed (i.e. set to zero). Instead of a periodic resetting of the throttle count to "I" as described in the paper, the throttle count is not reset at all. Instead, an upper limit is placed on the size of the throttle count. Also, the throttle count is now incremented by an amount "K" rather than by 1 when the test job is successfully executed. The parameter "V" (the time between checks) is set so that the probability of a successful test is about 0.5 at very high offered load.

Q.4 (Ralf Lehnert)

1. How is the throttle process activated - as a foreground job by a timer interrupt? or?
2. Transient behaviour: after or during a heavy overload situation: what do you do with calls in the input Buffer which experience (very) long waiting times? Is there a timeout mechanism?

A.4 (M. Eisenberg)

1. The throttle process is executed as a high-priority background process which is generated by a foreground timer interrupt at regular intervals. (Thus, the execution of the throttle process may be slightly delayed)
2. There is no time-out mechanism and no Input Buffer size limit. Calls are permitted to remain in the Input Buffer indefinitely, until they are served. In a sustained overload, some callers will ultimately abandon.