ABSTRACT

The subject of this paper is the automatic control of overloads in the Stored Program Control (SPC) of digital telephone exchanges.

Particular attention is placed in the choice of overload detectors and some methods of controlling the phenomena are compared with a simulation model.

An overload-control method based on the observation of the offered traffic has been chosen and refined to handle the maximum traffic load in normal conditions, yet ensuring an effective control of overloads.

1. INTRODUCTION

In a Stored Program Control system, the lack of network resources, storage capacity and elaboration potential may create overload conditions.

As the relative costs of the switching network and the memory devices diminish, it becomes economically convenient to exploit the bottleneck of the system (i.e. the elaboration devices) in its limiting operating conditions. The Central Control is, therefore, very sensible to overloads, since it concentrates all the most important activities, which may paralyse the whole system if they are not executed.

This has brought about a diffusion of methods which automatically control the overloads. They are generally based upon the limitation of the acceptance of new call requests to guarantee a complete overload protection, either internal or external, and at the same time ensuring maximum load handling in normal conditions.

Proteo is a fully electronic SPC system serving about 30,000 subscribers in a multinode configuration of local and transit exchanges (see [1], [3]).

This paper deals with a transit exchange configuration in which the interventions limiting the overload are exclusively competence of the Central Control. It is therefore necessary to define a maximum call-handling threshold, above which the carried traffic will decrease as a function of offered traffic.

Test trials have been effected on a simulation model, in order to choose the best overload detectors and specify the most efficient control procedures.

2. STRUCTURE OF THE SYSTEM

The Proteo Central Control (CC), consists of two special purpose computers, running in parallel synchronous mode. The twin computer complex works in a Master/Slave mode: both processors work synchronously and receive inputs from all connected peripherals, which accept commands only from Master processor.

The application programs are organised in priority levels which are executed according to a 10 msec. clock schedule.

The higher priority levels comprise the 'system activities': timing, alarm diagnosis, dialogue with other computers, etc. Some of these levels are activated by interrupts in preemptive resume mode.

The Call Handling (CH) level comprises all the traffic dependent tasks. All the incoming signals concerning the calls, generated by the subscribers or other devices in the system, are accepted in DMA in the CH buffer, and are analysed by the CH program thereby effecting the switching control operations.

The lowest levels are served only after the CH buffer has been emptied and are reserved for some necessary but deferrable tasks (cyclic tests, statistics, etc.). When these activities are complete, the processing unit (CPU) remains idle until the end of the cycle.

Fig. 1 - Software organisation
The response time in call handling is a function of CPU occupancy \( \rho \) for the 'primary activities':

\[ \rho = \rho_o + a \lambda \]

The term \( \rho_o \) comprises scheduling and monitoring tasks, and the high priority activities. In normal conditions this represents a constant load of approximately 10%. The second term represents the CH level activity, which for a given traffic mix is proportional to the offered call rate \( \lambda \). ('a' represents the average elaboration time for each call attempt).

An overload of elaboration may depend on either of the two tasks. An internal overload may depend upon the simultaneous arrival of a large quantity of alarms. An external overload is typically due to an excess traffic.

In overload conditions (\( \rho \) near to 1) some signals will be lost due to buffer overflow or an excess of response time. The system will operate at maximum load without serving any call with an acceptable response time.

3. OVERLOAD CONTROL

Short term overloads (less than 1 sec) may be absorbed by:
- exploiting the storage and elaboration reserves; the latter consists of the idle quota which in normal conditions is of 30%;
- deferring automatically the execution of less essential programs, with priority inferior to CH.

All the internal overloads and brief signal peaks are handled in this manner without service degradation. A lasting external overload will undoubtedly cause the collapse of the system and must be limited by a specific control mechanism.

The proposed intervention is to reduce the load by refusing new calls in order to ensure proper service to the calls already accepted and to a quota of the fresh ones. The refused requests are lost and cleared, because it is considered too onerous to store them for eventual elaboration.

A distinction should be made between the two following cases. In the case of a local switching system, the CC can delegate the filtering of fresh requests to the decentralised organs. This paper, on the other hand, deals with the more complex case of a transit exchange between traditional switching systems; the signalling interfaces between junctions and control feed directly the CH buffer with signals, without effecting any selection or filtering.

Only the CH program is able to distinguish the signals and eventually refuse the new calls. This centralised operation permits a selective choice discriminating between subscribers and incoming trunks.

When the call is not accepted, the elaboration load to recognize and release it constitutes an overhead reducing the efficiency of the control intervention.

4. CONTROL OBJECTIVE

The control objective is to limit the occupancy \( \rho \) to a safety value \( \rho_s \) which nevertheless ensures the quality of service requested (fig 2 a, b):

\[ \rho \leq \rho_s = 0.9 \] (4.1)

corresponding to a traffic limit:

\[ \lambda_s = (\rho_s - \rho_c) / a \]

This constitutes an intervention threshold above which the traffic load will be limited by accepting only a quota \( \lambda_c \lambda \) of the offered traffic.

Since each request refused will occupy the CPU for a time \( b = g a (\rho \leq 1) \)

\[ \rho = \rho_o + a \lambda_c + b (\lambda - \lambda_c) \] (4.2)

or:

\[ \rho = \rho_o + a(1-\rho)\lambda_c + a\lambda \]
The control condition (4.1) determines the limiting value of the carried traffic:

\[ \lambda_c = \frac{p_r \cdot \lambda_c}{a(1-p)} - \frac{\theta}{p'} \lambda \]

or:

\[ \lambda_c = \frac{\lambda_s}{1-p} - \frac{\theta}{p'} \lambda \]  

(4.3)

With the intervention threshold in mind, it is possible to define a 'control function', offered traffic / carried traffic, which is to be considered an objective in any overload condition (fig. 2 c , with \( p = 0.16 \)):

\[ \lambda_c = \frac{\lambda_s}{1-p} - \frac{\theta}{p'} (\lambda - \lambda_s) \lambda_s \lambda_c \]  

(4.4)

The excessive work involved in refusing the calls is the reason whereby the traffic that can be carried decreases, as the offered load increases.

5. INFLUENCE OF REPEATED-CALL ATTEMPTS

As a consequence of call refusal, repeated-call attempts will constitute an incremental factor of the overload.

Fortunately, the traffic-increase phenomena is spread in time due to the effective interarrivals of the reattempts.

As an initial approach, the hypothesis on the first model of Le Gall are assumed (interarrival between successive reattempts >> holding time). In equilibrium and with a constant repetition factor \( \alpha \) for the refused requests, the total traffic may be decomposed into fresh traffic \( \lambda_s \) and repeated calls:

\[ \lambda = \lambda_s + \alpha (\lambda - \lambda_c) \]

The above relations may be rewritten in terms of fresh offered traffic and are given in fig. 3 with \( \alpha = 0.5 \).

6. OVERLOAD DETECTORS

An objective of this study was to determine the overload detectors or system variables which, correctly measured, revealed in time the advent of overloads.

The main requirements were:

a) efficiency, with minimum overhead due to measurement;

b) significant relation with controlled variable (computer occupancy);

c) correctness, by discriminating between dangerous overloads and momentaneous traffic peaks in order to reduce the useless interventions;

d) timeliness, in intervention

e) proportionality, to determine gravity of overload and graduate the intervention in accordance with the objective function.

In view of the properties of the system, three parameters which respond to the requirements (a) and (b) have been selected:

- CPU occupancy \( \gamma \), utilising as detector the actual variable to be controlled;
- queue-length of the buffer \( CH \), which is a function of \( \gamma \);
- offered traffic intensity

The overload-control mechanisms based upon the measurement of the above values, were compared with the intent of meeting the requirements (c), (d) and (e).

In all the above mechanisms the actions, measurement and control, evolve in cycles of duration \( t \):

- The measurement is a average throughout the duration of the control cycle \( t \); this technique aims to satisfy the requirement (c) by smoothing the variations of brief intervals. A useful compromise between the requirements (c) and (d) is ensured by taking \( t = 0.5 \) sec., value which is not critical.

- At the end of each cycle, an algorithm determines the traffic accepted \( \lambda_c \) for the successive cycle.

- During the cycle, the program \( CH \) accepts the new requests up to the relative quota 'Call Acceptance Limit' (CAL = \( \gamma \lambda_c \)).

The fact that the acceptance of the requests occurs mainly at the beginning of the cycle, has not produced any negative effects.

7. SIMULATION TRIALS

The simulation trials were effected by means of a model, written in Simula 67, reproducing the software structure and the management strategy of the system. The model implemented the different overload control algorithms in order to compare them.

The input traffic was reproduced in detail as a subscriber process generating a succession of...
signal messages in the course of time. The originating call-mix was generated by a Poisson process and the issue of each call determined a priori.

Repeated call attempts were simulated with a retrial factor \( \alpha = 0.7 \) after a mean time of 30 sec. The results are presented in terms of total offered traffic, and independent of the repeated-call component.

The overload-control mechanisms were compared through the main system variables (CPU utilization, response time, queue length, carried traffic) in terms of mean values and in time. Some significant distributions have also been measured.

The response of the system was simulated with overloads lasting 3+4 minutes. The statistical measurements were effected in the last 3/5 of this period and involved about 70,000 + 90,000 signals.

In the following are described the control algorithms employing the detectors previously described, and the relative simulation results are presented.

8. MEASUREMENT OF CPU UTILISATION

This is a measurement of the parameter to be controlled.

It is effected without overhead by measuring the idle-states with a dummy sampling process. The estimate of \( \varphi \) will include two types of error which may be neglected:

- fluctuations of less essential activities, which should not appear in the term \( \varphi \);
- scheduling times of the idle process: the process sees the CPU virtually dedicated and is not able to count the number of times, in the measurement cycle, that it has been activated or suspended. It is, therefore, not possible to measure the total loading required by its scheduling.

This measurement does not determine the proportion of overload since the loading is limited to 1 for any value of overload.

In the trial mechanism, the control algorithm determined the acceptance limit according to a threshold table:

<table>
<thead>
<tr>
<th>( \varphi )</th>
<th>( \lambda_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq 0.92 )</td>
<td>1.20 ( \lambda_s )</td>
</tr>
<tr>
<td>&gt; 0.92</td>
<td>0.88 ( \lambda_s )</td>
</tr>
<tr>
<td>&gt; 0.94</td>
<td>0.56 ( \lambda_s )</td>
</tr>
<tr>
<td>&gt; 0.98</td>
<td>0.28 ( \lambda_s )</td>
</tr>
</tbody>
</table>

The choice of different thresholds or the institution of separate on-off thresholds for each value of \( \lambda_c \) did not sensibly modify the results.

The default limit which was always present, constituted an attempt to stem sudden traffic in-

creases in view of the successive intervention of the overload control.

In effect, it appears that the drawback of this method is its lack of timeliness.

The elaboration load required by a new call, spreads over the entire selection phase, producing a delay between the traffic variation and the loading it produces. This effect is clearly shown by examining the measured variables, in time, of the system simulated with the overload control here described (fig. 4).

The control intervenes (due to the increase in CPU loading) in the points marked \( \circ \), by reducing the accepted call requests. The benefit (or reduction in CPU load) manifests itself after many seconds. In the mean time, the system will remain in overload state and the queue tend to explode.

Since the detection mechanism is based on a consequence (CPU load) which is delayed respect to the intervention variable (traffic), the control is always late at the beginning and at the end of each intervention.

This provokes some oscillations in the system which passes from moments of overload with service degradation to phases of scarce utilisation. The result produced, also in its mean value, will be of only a slight improvement in service respect to a system without control (curve R in fig.6a) although the 'control function' offered traffic carried traffic is met (fig.6c).

![Fig. 4 - Time evolution with \( \varphi \) - control](image-url)
9. MEASUREMENT OF QUEUE LENGTH

The queue length of the CH buffer is sampled every 10 msec. at the end of the CH service, and a mean is obtained through the measurement cycle. This variable is strictly related to the CPU load but respects more correctly the requirement of proportionality. On the other hand its measurement overhead is greater.

The algorithm tested, was organised in thresholds just as the previous one:

<table>
<thead>
<tr>
<th>Queue Length</th>
<th>( \lambda_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq 2 )</td>
<td>( 1.20 \lambda_s )</td>
</tr>
<tr>
<td>&gt; 2</td>
<td>( 0.88 \lambda_s )</td>
</tr>
<tr>
<td>&gt; 10</td>
<td>( 0.56 \lambda_s )</td>
</tr>
<tr>
<td>&gt; 100</td>
<td>( 0.28 \lambda_s )</td>
</tr>
</tbody>
</table>

The time evolution graphs in effect show more graduality of intervention (fig. 5). But the lack of timeliness, and the consequent oscillations, remain the same.

The resulting overhead is minimal, whilst proportionality is guaranteed. Since this consists in a direct measurement of a possible overload factor, the timeliness of intervention is ensured.

The control algorithm acts as follows:
- the offered traffic \( \lambda_s \), determined in the previous measurement cycle, is assumed for the successive control phase;
- the limit \( \lambda_c \) of carried traffic is directly obtained from control function (4.3):

\[
\lambda_c = C - B \lambda
\]

where

\[
C = \frac{\lambda_s}{1 - \beta} \quad B = \frac{\beta}{1 - \beta}
\]

The very nature of the algorithm ensures the graduality of intervention.

Simulation trials (figs. 6 and 7) have shown that...
the timeliness is sufficient, no oscillation phenomena were revealed and the grade of service (response time) is quite constant as the overload varies.

The mechanism described above smooths the accepted traffic, with respect to theoretical objective, around the threshold value \( \lambda_s \) (curve T, fig. 9). In this region the number of new requests per cycle approximates on average the value CAL, although this value is not reached in all cycles due to statistical variations.

An improvement is obtained by modifying the control function according to the traffic regions (fig. 8):

11. IMPROVEMENT OF THE ALGORITHM

The overload region (I) follows the (4.3) control function; the transition region (II) around the value \( \lambda_s \) increases the virtual acceptance limit to compensate the above effect;

the region of normal load (III) foresees a constant safety limit.

Since the traffic estimate should be precise without oscillations, the measure \( \lambda^{(k)} \) of the k-th cycle will be the weighted average of the previous measurements, \( \theta \) being the smoothing factor:

\[
\lambda^{(\infty)} = \theta \lambda^{(k)} + (1-\theta) \lambda^{(k-1)}
\]

This traffic estimate determines the acceptance limit for the cycle k+1.

The adjustment of the control function and the choice of the coefficient \( \theta \) were determined with a simplified simulation model. The optimal choice is with function shown in fig. 8 and a value of \( \theta = 0.2 \).

The corresponding simulation results with the main model, appear in fig. 9, curve \( T_1 \), and approximate the control objective in a satisfactory manner.

The efficiency of the offered-traffic control mechanism requires a correct evaluation of the load parameters 'a', 'b' of (4.2):

\[
\phi = \phi_c + a \lambda_c + b (\lambda - \lambda_c)
\]

from which the control function is determined in the overload region.

The elaboration load 'b' for unaccepted requests is defined. The load due to accepted requests 'a', on the other hand, is a function of traffic mix and related with the following variables:

- destination (number of digits selected);
- result (incomplete or without response attempts).

These variables, in particular the second, depend upon the hour, the state of the network, etc. Fortunately the value calculated for an average traffic mix in normal conditions, is conservative for those critical conditions when an increase in unsuccessful call-attempts is expected.

There is however a risk of using the system below the value \( \phi_c \).
A method is under study whereby the comparison of the measured and the theoretical utilization will permit an adjustment of the parameters of the control function.

This action should occur with intervals of 15+30 sec. in order to avoid the risk of oscillations connected with the measurement of $f$.

12. CONCLUSIONS

The simulation trials have shown that the choice of overload detectors significantly affect the performance of the overload-control methods.

Observing the computer utilization proved to be an insufficient measure to control overloads quickly enough; oscillations in the service provided, resulted in a loss of quality.

The periodic observation of offered traffic, on the other hand, quickly revealed the imminent origination of external overloads. The measure was easily implemented by an algorithm which limited the calls to be accepted, maintaining thereby constant the computer utilization and consequently the quality of service.

The control mechanism functioned properly also in conditions of extreme overload, and has been improved to ensure the specified service in normal conditions or near the preset threshold limit of intervention.

REFERENCES