Copyright © 1982 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 61, No. 6, July-August 1982
Printed in U.S.A.

Automated Repair Service Bureau:

The Context-Sensitive Switch of the Loop Maintenance Operations System

By J. P. HOLTMAN

(Manuscript received June 5, 1981)

In a distributed system where the data base is partitioned, e.g., on a geographical basis, incoming transactions must be routed to the correct computer based on information in the data. In some instances, this is performed manually by the attendant entering the data, but as the switching criteria becomes more complex and the transaction load increases, a mechanized system which could examine the data and route automatically would decrease the time required for an attendant to handle an individual input and, thereby, increase the productivity of the attendant. This paper describes the design and implementation of a "context-sensitive" switch that is used to route transactions in the Loop Maintenance Operations System.

I. INTRODUCTION

The Loop Maintenance Operations System (LMOS)[1] is a hierarchical network composed of a host computer[2] which is a large data base machine, and a number of transaction processing front-end (FE) computers.[3] The entire data base is contained on the host computer, but this data is distributed to each of the FE computers based on the geography of the system. The geographical boundaries that are defined by an FE are those areas served by a Repair Service Bureau (RSB).

Since an RSB typically is associated with several wire centers, the customer lines that are served by that bureau correspond to the geographical area served by those wire centers. These lines are typically grouped into what are referred to as NNXs. An NNX is the first three digits of a telephone number, commonly referred to as an exchange. When LMOS was first designed, all trouble reports for a given geographical area were routed to the appropriate RSB automatically by the telephone switching equipment. There, the attendant would input the trouble report to the FE that served that RSB. In this case, everything worked fine as long as the calls were routed to the correct RSBS.

As the Bell operating companies (BOCS) consolidated work centers to give better service, and reduced the size of the staff necessary to do that function, they created the Centralized Repair Service Attendant Bureau (CRSAB). This is the central site that handles all the trouble report calls for a very large area; this area now spans many FES. Since the calls are answered in the CRSAB on a random basis, there is no longer the guarantee that the calls will be directed to an attendant who has a terminal that is directly attached to the proper FE. Therefore, the attendant has to have the capability of dynamic switching his/her terminal between any of the FES to input the trouble reports.

In an existing BOC, there were six FES servicing the entire metropolitan area. When the CRSAB was created in this BOC, it necessitated the capability for each answering position in the CRSAB to be able to access any one of the six FES. This implied that the attendant, upon taking the trouble report, had to determine which of those six FES contained the telephone number referred to in the trouble report. Therefore, the attendant had to be able to determine for a given NNX (one out of 600 in this area) which FE contained that NNX (one out of six).

This was exactly the manual solution that the BOC was forced to take. At each of the answering positions in the CRSAB, there was located a six-position switch. This allowed the attendant to select one of the six FES in which to input the trouble. The appropriate FE was selected on the basis of the telephone number for which the trouble was being reported. A table was provided giving the FE associated with an NNX and the attendant would consult this to determine the FE.

If the NNX was not in the table, the attendant would try to enter the transaction on each of the FES until the transaction was accepted or negative responses were obtained from all the systems. If all the responses were negative, the attendant would change the transaction name to route this trouble for special handling. This manual switching arrangement was expensive and complex. It was also dependent upon a specific type of terminal and, therefore, was not usable by other BOCS.

To provide a system that was usable by all the BOCS and to provide for future extensions to the LMOS system, the design of the Cross Front End (XFE) was undertaken. The main function of the XFE is to automatically determine the FE that contains a particular phone number. When this determination is made, it routes the input trans-

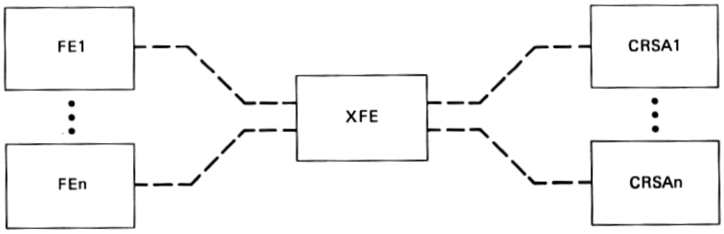

Fig. 1—Cross front end in the LMOS network.

action to that FE, and when the response to that transaction is obtained, it routes the response back to the appropriate terminal. The XFE is situated in the network in such a way that all the terminals in the CRSAB are connected to the XFE, and the XFE, in turn, is connected to all the FEs. Therefore, the XFE appears in the same location as the manual switch did in the previous discussion and serves the same purpose, except now the function is being done automatically.

II. BASIC DESIGN OF THE CROSS FRONT END

The basic design for the XFE was relatively simple. Its function was to be a "context-sensitive" switch. That is, it would look at the phone number on a transaction, look that phone number up in an internal table which gave the mapping between phone number and FE, and then route the transaction to the FE indicated. When the response to that transaction was obtained from the FE, its function was then to route that response back to the terminal that originated the transaction.

Since a BOC could optionally install an XFE as part of an LMOS system, it was designed so that it did not affect the operation of the FE software. That is, it had to be completely transparent to any existing software in the LMOS system. Therefore, the XFE was designed to appear as a terminal controller to the FE, making it transparent to the FE, whether there was a real terminal controller at the other end of the line, or an XFE simulating the terminal controller. (See Fig. 1.)

The hardware configuration for the XFE consisted of a PDP* 11/34 computer with 256 kilobytes of memory, a cache memory to increase the speed of the processor, a maximum of 32-line controllers, and a floppy disk for loading the system software and for recording system dumps on abnormal terminations. For reliability reasons, the XFE was configured in either a duplex or a triplex configuration. A duplex consisted of a live XFE and a backup XFE. A triplex configuration consisted of two live XFEs and a backup which could be configured as

* Registered trademark of Digital Equipment Corporation.

either one of the other XFEs. Switching between a live and the backup was accomplished with a manual switch and the entire operation was initiated by the operators in the computer center.

III. STRUCTURE OF THE XFE SOFTWARE

The XFE software consisted of three major components:

(*i*) The operating system
(*ii*) The communications management software
(*iii*) The application code.

The operating system used for the XFE was the Bell Operating System (BOS11); it is the same one that was used on the FE computers. The BOS11 was field proven, reliable, and very few changes were made in it except to include the new device drivers for the communication lines.

The Communications Manager on the XFE was completely rewritten from the Communications Manager that existed on the FEs. The reason for this is that the Communications Manager on the FEs was not general enough to handle the capabilities required in the XFE.

The Application Code (APL) is the software that implements the functions of the XFE. When a transaction is input, APL uses information contained within the input data to switch that transaction to the appropriate FE. The APL divided into three modules.

(*i*) A module to handle the interactions with a terminal
(*ii*) A module to handle the interactions with a FE.
(*iii*) A module to handle the interactions with the XFE console terminal.

The rest of this section will discuss the design of the XFE in more detail.

3.1 System configuration

An XFE can handle a maximum of 150 terminals, while an FE has the capacity for 512 terminals. If the XFE were configured in the simplest fashion, then each terminal on the XFE would have an appearance on an FE. This would allocate 150 of the 512 terminals on an FE to the lines serving the XFE. But, at any one time, only a small fraction of the terminals on the XFE would be in communication with a specific FE. Also, since the customer contact time (120 s) is long compared to the response time (10 s) of a transaction, the XFE was designed to dynamically couple a terminal to an FE only for the duration of the transaction. A transaction is defined as the input from a terminal and the output response returned to that specific terminal. Therefore, after the output has been sent back to the specific terminal, the XFE can free up that coupling for use by another terminal. This allowed for the optimal use of the resources both by the XFE and the FEs.

When the XFE was put into the network, there were two physical communication lines from each FE to the XFE. These communication lines were duplexed for reliability. A single communication line was capable of handling the necessary transaction load between the FE and the XFE. On each of these communication lines, there was the appearance of 16 pseudo terminals (PTERM). When the XFE receives an input transaction from a real terminal, it dynamically allocates an available PTERM on the FE to which the transaction is to be routed. The XFE then transmits the transaction so that it appears, to the FE, as if it were entered from the PTERM. When the response is received at the XFE from this PTERM, it is routed back to the original input terminal and the PTERM is deallocated, freeing it for another conversation between the XFE and the FE.

3.2 Routing definition tables

The routing tables provide the XFE with the information necessary to determine how to route the transactions to the appropriate FEs. These routing tables consist of two basic sets of tables. The first set of tables contains the transaction name and the location of the field that the XFE is to use to obtain its routing information. The other set of routing tables contains the routing criteria; i.e., a list of NNXs versus FEs that contain those NNXs.

The information for these routing tables is obtained from two sources. The table that contains the information on the transactions is prepared manually by the operations personnel at the BOC from information provided by the Western Electric Company. The information contained in the routing criteria tables is created automatically from information on the FE computers. An off-line program is run on each of the FE computers in an LMOS installation that reads the data base on the FE computers to determine which NNXs are loaded on that data base. The results of this program are recorded on a tape, and the tapes from all the FEs are merged on to one single disk from which this table is generated. Once these tables have been generated, they are loaded on the XFE floppy disk and automatically loaded into memory when the XFE system is booted.

3.3 Network definition tables

Each XFE installation has its own unique requirements in the way that the physical communication lines are configured. As the system grows, an XFE may initially be connected to two FEs and finally to six FEs. At each stage, the XFE must know the actual communication network configuration attached to it. To do this, a facility is provided so the personnel in the computer center can describe the configuration to the XFE.

The description of the network configuration is prepared on a card deck that is read by an off-line utility program which stores the resultant tables on a floppy disk file that the XFE can read when the system is initialized.

3.4 Processing special service numbers

So far, it has been assumed that the XFE uses the telephone number that was input to determine its routing criteria. This routing was done by looking up the first three digits of this phone number to determine to which FE to route transaction. For most POTS ("plain old telephone service") numbers, this assumption is true. And since POTS numbers make up a large percentage of the subscriber lines on which troubles are reported, then this represents normal processing required on the XFE.

But, customer circuits, called Special Service Circuits, exist that do not have the normal seven-digit phone number that is typically associated with a telephone. For example, if a customer has a dedicated data line connecting two computers in different locations, this line may be designated with a phone number that would look like "96PL1234A." If one tried to use the first three digits of this phone number, "96L," to determine the routing criteria, the XFE would determine that this NNX does not exist in any of its tables.

If this were the manual system, the attendant would first try to input this transaction to FE number one, get the response, and if the response said that this number did not exist on FE number one, the attendant would switch to FE number two and repeat this scenario until either the FE was located that contained this number, or all the FEs had been tried to no avail. Therefore, it was required that the XFEs somehow implement this same capability.

To handle Special Service numbers, an "inquiry" transaction was built into the XFE. In the case where the XFE could not determine the routing criteria by doing a table look-up on the data, the XFE would initiate an "inquiry" transaction simultaneously to each FE to which it was attached. This inquiry transaction consisted of the phone number for which the XFE was trying to determine the routing. On the FE, the inquiry transaction would attempt to access the data base with this phone number. If the data did exist on the FE, the transaction responded with a positive acknowledgment. If the data did not exist on that FE, the FE responded with a negative acknowledgment.

When a positive acknowledgment is received, the XFE would route the transaction to that FE. If no positive acknowledgments were received from any of the FEs, the transaction would be routed to the last FE to respond. This was done so that the XFE did not have to generate any error message on its own. It would route the transaction to an FE and the application program on the FE would be written such that if it did not find the data it needed, it responded with a known error message with which attendants were familiar.

One by-product of this capability of making inquiries took care of the case where a BOC did not update the XFE routing tables when a new RSB was added, or when a new set of phone numbers were added to an FE. Since this information would not have been contained in the XFE routing tables, when one of these new phone numbers was entered on a transaction, the XFE would automatically generate an inquiry transaction and the FE containing that phone number would send a positive acknowledgment.

3.5 Operator control

The XFE was designed as a standalone system; i.e., it did not require any operator interaction. If a system failure occurred, a dump was taken automatically to the floppy disk, and the system rebooted itself and started up again. Because of the limited resources on the XFE (memory and off-line storage), the console terminal was used both as an interface to the operator and as a logging device. The information logged on the console terminal consisted of the number of transactions entered, average response time for those transactions, and other miscellaneous data that could be used both to determine the performance of the system, and to analyze any strange behavior of the system.

At the console terminal, the operator could control which lines were active, which terminals were active, and change some of the system parameters to tune the response of the system. The XFE also allowed any operator command to be input from the system control terminal on the FE. The operator would input a special transaction on the FE and the application program that was executed as the result of this transaction would output the data (command) on the communication line between the FE and the XFE. When the input was received at the XFE, it was interpreted as if it came from the console terminal. Responses to that command appeared both at the XFE console terminal and as a response sent back from the XFE to the FE so that it appeared at the user's terminal. This allowed the operators to control and monitor the XFE performance without having to use the terminal at the XFE.

IV. PUTTING THE SYSTEM INTO THE FIELD

There were adequate test facilities in the laboratory for both the development and for the testing of the software. But the real test came in trying to put this software into an actual installation in the field. It must be remembered that CRSAB is a very critical work center in the LMOS system; it takes the initial trouble report and initiates all the basic functions that LMOS performs. Therefore, if service is discontinued for any reason to the CRSAB, there is a severe impact on the LMOS system.

As was previously mentioned, the BOC where the XFE field trial was conducted had a manual system for switching the terminals between the various FE computers. It turned out that the manual switch that was used at each of the operator positions actually was an eight-position switch. When the field trial started, the XFE was attached to one of these extra two positions on the switch. Therefore, the operator could manually select one of the six FEs as they originally did, or for the field trial, select the XFE. This allowed for parallel operations during the field trial.

Initially, 16 CRSAB positions were attached to the XFE so that system response could be gauged to a small number of users (the XFE was designed to handle 150 terminals). Since the attendants could switch back to the manual system at any time, this provided feedback on the system performance by monitoring how often these positions had to resort to manual operations because of response time problems.

4.1 Handling slow responses from a front-end

The normal response time at a terminal not connected to the XFE was 3 to 5 seconds, and it was anticipated that the XFE would add a second to this response, bringing the expected response with the XFE to the 4- to 6-s range. Early results in the field trial indicated that response time for terminals connected to the XFE was in the 10- to 13-s range. These results were not satisfactory and, therefore, an effort was undertaken to determine the bottlenecks in the system.

At this point, statistics built into the code came in very handy. By analyzing the output and displaying some internal tables on the console terminal, the problems causing this slow response time were identified.

One of the major problems causing slow response time had not been anticipated in the design phase. This had to do with detecting when one of the FEs connected to the XFE was "down"; i.e., the FE was unable to accept the transactions that the XFE was attempting to send it. The reasons for this could be either

(*i*) The FE was physically down because of some hardware or software problem, or

(*ii*) The FE was heavily loaded and responding very slowly to transactions that were sent to it.

Whatever the reason for this slow response, it caused a longer holding time for the system buffers on the XFE. In the initial design phase of the XFE, it was anticipated that a buffer would be held for a relatively short period of time—2 to 4 seconds. This was the time it took to send the transaction over to the FE and get a positive acknowledgment that the transaction was received. When an FE goes down, it is not sending back this positive acknowledgment and therefore is tying up the system buffer on the XFE.

Now the XFE had a time-out such that after approximately 30 s it would release this buffer and send a message back to the user's terminal indicating that the transaction could not be completed. But, if a large number of transactions had been input which were destined to this FE, then a large number of buffers would be tied up for a long period of time. Since buffers are one of the scarce resources on the XFE end, this would lead to a bottleneck in the system, preventing the transactions going to other FEs.

To overcome this problem, a system parameter was defined which specified the maximum percentage of the available buffer space that would be allocated for transactions that were destined for a specific FE. By default, this value was set at 30 percent, so if an FE was slow in responding, only 30 percent of the available buffer space would be allocated for use by that FE. If a subsequent transaction were entered that was to be routed to that FE an error message would be returned to the user indicating that no buffers were available and that the user should wait a few seconds and then reenter the transaction.

If it was determined that an FE was physically unavailable, then the percentage of available buffer space that it was allowed would be dynamically changed to zero, preventing any transactions that would be routed to that FE from being entered into the system. When the FE came on-line again, its percentage would be changed back to the system default. With these changes, the desired response of 4 to 6 seconds was achieved.

4.2 Buffer management

The scarcest resource on the XFE is the memory space available for buffers. The XFE places a large demand on this resource because of the number of conversations that the XFE can have in progress at any one time, and because of the requirements of the bisynchronous protocol.[4] The limited address space on the PDP-11 (a program can have a total of 64 kilobytes of text and data under BOS11) required that the buffers be kept in a separate address space from the programs using them. System calls were provided to allocate/free the buffers, but because of the heavy usage of this resource, 10 percent of the CPU was used to service these requests.

To improve the performance of the system, a "cache" of buffers (actually the address descriptors of the buffers in the separate address space) was maintained in the application program. Therefore, if the application program required a buffer, it would first look in this cache to see if there was an available buffer descriptor. If there was, this buffer descriptor was used, saving a system call. When a buffer was freed, the application program would look for an empty slot in the cache, and if there was an empty slot, place this buffer descriptor in it.

If there was no available buffer descriptor in this cache when the application code requested one, a system call was made to allocate this buffer. But instead of allocating a single buffer, the system would allocate five buffers. One was returned to the application program and the other four were used to populate the cache. This was done in anticipation of future calls requesting the allocation of buffers. This cache algorithm allowed over 80 percent of the requests to be satisfied without requiring a system call.

The bisynchronous protocol required that a "large" buffer (3000 bytes) be allocated any time data was expected from an FE. Since there was a limited number of these "large buffers," and because the average size of the response from an FE was on the order of 200 to 300 bytes, a function was added to the system to "shrink" the data from a "large" buffer into a "small" buffer of the correct size; thus, freeing up this large buffer for subsequent input.

Detailed statistics were maintained on the buffer utilization and these statistics were printed out periodically on the console terminal. This allowed the developers to monitor the buffer usage of the system, and it also gave information necessary to track down any anomalies that may have occurred.

4.3 The communications manager

The communications network managed by the XFE is referred to as a "multipoint bisynch"; i.e., the lines in this network must be polled to see if a terminal on a line has data for input. To give reasonable response time to the terminal, the line must be polled approximately once every half second. On a typical XFE, there are 10 to 18 bisynchronous lines.

Each poll requires two I/O operations, and a system call is required to initiate an I/O operations. This system call requires approximately 5 ms. Calculating the time required to handle 18 synchronous lines, there are:

$$18 \text{ lines} * 2 \text{ I/Os/poll} * 2 \text{ polls/second} * 5 \text{ ms/I/O}.$$

This equals 360 ms/s or approximately 36 percent of the CPU just to support the communication lines with no other activity in the system. Under load with all the other functions in the XFE must handle this overhead is intolerable.

When an analysis was done of what happens during the polling, it was determined that 60 to 80 percent were "idle" polls; i.e., there was no data input from the line. Therefore, a method had to be found to reduce the overhead of these nonproductive polls.

One way of doing this was to put the polling capability at the lowest level in the system; i.e., at the device driver level in the operating system. Also, the driver can determine if it was an "idle" poll and if so, just initiate the next poll 0.5 s later. By putting this "auto-poll" capability in the device driver, the overhead associated with the poll was reduced from 10 ms (two I/Os times 5 ms) to 1 ms. This was a substantial reduction in the overhead required for the communications manager.

A similar capability was also required on the lines connecting the FE and the XFE. Because the XFE appears as a terminal controller to the FE, the FE polls the XFE to determine if there are any data to be transmitted. If the XFE does not have any data ready for transmission to the FE, it must respond with an "eot" and set-up to receive the next poll. If this I/O is being done from the user program, this would require two I/Os (or a total of 10 ms) to answer each of these polls. By moving this function down into the device driver again, the overhead is significantly reduced to 1 ms to answer a poll from the FE. This capability is referred to as the "auto-answer" capability.

V. PERFORMANCE

The XFE is designed to handle a maximum of thirty-two 9.6-kilobaud communication lines. In its maximum configuration, the lines would be allocated as follows:

Lines	Function
2	LMOS host computer
20	Two lines each to a maximum of 10 FEs
10	Terminal controllers (typically 16 terminals)

There are currently over 30 XFE systems deployed (and more being added) with each XFE switching a maximum of 5000 transactions per hour.

VI. CONCLUSION

The XFE was initially developed to handle the switching of transactions in a CRSAB. It was thought that only a few of the BOCs would select this optional capability because many of the smaller BOCs could probably get by with a manual switch. But as LMOS evolved and the XFE proved to be a reliable system in the field, there was a need for this switching function in other work centers besides the CRSAB.

As the BOCs desired more monitoring capabilities, some of the operations staff terminals were attached to an XFE. Also, since the XFE acts as a terminal concentrator, and in many BOCs there is a shortage of lines on the FEs, the XFE can be used to increase the available number of ports to the LMOS system. This is especially true when the BOC wishes to add some low usage terminals that do not require fast access to the system.

VII. ACKNOWLEDGMENTS

I would like to thank the people who are responsible for the software in the XFE. B. B. Bittner and W. F. Hoyt designed the application code that implemented the switching functions of the XFE. J. P. Haggerty, S. Myer, and R. W. Underwood developed the communications management software that allowed the XFE to "talk" to the outside world. J. Cloutier made the changes in the BOS11 to support the XFE and helped to tune the system. Without their help, and the long hours spent in the development and field trial phases, this paper would not have been written.

REFERENCES

1. R. L. Martin, "Automated Repair Service Bureau: System Architecture," B.S.T.J., this issue.
2. C. M. Franklin and J. F. Vogler, "Automated Repair Service Bureau: Data Base System," B.S.T.J., this issue.
3. S. G. Chappell, F. H. Henig, and D. S. Watson, "Automated Repair Service Bureau: The Front-End System," B.S.T.J., this issue.
4. "IBM 3270 Information Display System Component Description," IBM Publications GA27-2749-9, August 1979.