A SIMULATION MODEL FOR A LOOP ARCHITECTURE DISTRIBUTED COMPUTER SYSTEM

S. Baragli and S. Valvo
Ing. C. Olivetti e C .
Progetto Centrale Sistemi
IVREA (TO)

ABSTRACT

Moving from a practical problem, the preliminary definition of a loop network, a simulation model has been built and will be described in this paper. For its characteristics of flexibility, modularity and programmability, this model has been used also as a tool for documenting the evolving project and as a design aid.

At the end, some of the results obtained are presented in graphical form: the loop utilization and performance at different transmission speeds, and the system's response time versus load variations.

INTRODUCTION

This paper is concerned with a simulation model that has been built as an aid for the project of a loop-connected distributed computer system for office automation applications.

Aloop network can be built between N terminals by systematically connecting the output of one to the input of the next and the output of the last to the input of the first (FRA74). In the case studied, it has been decided to let messages flow in only one direction (e. g. clockwise).

Many processors (the hosts) use this facility to communicate; they are attached to the loop through special hardware interface modules, called nodes. Any node, if not in transmitting mode, must always receive bytes from one side and retransmit them to the other, thus permitting messages circulation.

As two or more hosts could try to put messages on the line at the same time, some form of loop control must be provided to prevent mutual interference. This control can be either centralized (transmission by any node must be authorized by a special unit, the loop controller) or distributed. In this project, distributed control has been chosen for reliability and cost considerations.

At least two control strategies are possible:

a) when a node wants to transmit, it inserts the message between two other messages, delaying the second for as much time as necessary (REA76);

b) a particular "go-ahead" message circulates around the loop, allowing only one node at a time to trnsmit (FAR72, NEW69).

In the first case, a hardware mechanism must be provided to switch the incoming message in a delay buffer when so needed; the second strategy was deemed more simple and was adopted for this project. So, when a node has to trnsmit a message, it must follow the sequence:

i) wait for the "go-ahead" item to arrive;

ii) destroy this item and transmit the message instead of the item;

iii) transmit another "go-ahead" token, thus releasing the line.

In loop networks, the amount of time by which any message is delayed is called node delay. In the case studied, the node delay must be fixed in time; for the different trials simulated with the model, this delay has been made equivalent to the transmission time of one or more bytes.

At the software level, communication is process-oriented; there is only one set of communication primitives between processes, whether the processes are allocated on the same processor or not. To give an example, let us suppose that the process A, allocated on host n. 1, wants to send a message to the process B, that belongs to the host n. 2. The following actions will take place:

1) the interprocess communication facilities in the o. s. nucleus of host n. 1 determine, in a transparent way, whether the process B is local to the host n. 1;

2) as process B is not local, a command is issued to the node n. 1, in order to send the message through the loop;

3) if the process B is being addressed for the first time, the node has no way of determining that the process B is on host n. 2; so the message is sent on the line with the address 00 (broadcast message);

4) every node in the loop receives the message; obviously, only the node n. 2 passes it to its host; afterwards, there will be a reply message, containing the physical address of its sender (n. 2);

5) as the reply message reaches the node n. 1, this node becomes informed that the process B is allocated on the host n. 2; this information will be used, if necessary, by the node n. 1, but not by process A. This makes easier the reconfiguration of the system.

At a first approach to the definition of the real system, the need arose for an estimate of the minimum line transmission speed and consequently of the technology required for the interface transmitters. It was decided to carry out this estimate using a simulation model, written in the language SIMULA67 (BIR73). This model has not been used for this estimate only, but it grew with the project.

A project's development usually consists of:

i) the choice of the architecture to be utilized;

ii) the logical partitioning of the system into different modules, with the definition of the respective interfaces;

iii) the implementation of these modules.

The simulation model can be present in this process if it represents, at an high abstraction level, all of the system's parts. Then it can:

a) compare the performances of different system's architectures;

b) verify the consistency of the interface definitions between the various system parts;

c) provide a context to test an already implemented module; then cycle back to point b) and re-evaluate the throughput if this module behaves somehow differently from the specifications;

d) suggest, as a consequence of points b) and c), structure modifications in order to maintain or enhance performance;

e) document the project evolution.

The system studied has been designed for office automation applications: typical hosts are interactive intelligent terminals, printers, "smart copiers" etc.

Microprocessors will be used both for the hosts and the nodes, as they are low cost components and there is no need for high calculating speed (the results obtained show that the required line speed is of the order of 19200 baud).

THE SIMULATION MODEL.

The modelling facilities.

The simulation model has been implemented with a program written in the language SIMULA67, running on an IBM 370/168. We have taken advantage of the various modelling facilities included in the system-defined classes SIMSET and SIMULATION.

The class SIMSET allows the definition of circular two-way lists and of the associated elements, that can be manipulated by special primitives, such as INTO, OUT (to insert or remove the last member of the queue), CARDINAL (to obtain the number of elements in the queue) etc.

The necessary concepts for discrete event modelling are provided in the system class SIMULATION which is itself prefixed by SIMSET so that all the latter concepts are available. A simulation program is composed of a collection of processes which undergo scheduled and unscheduled phases. When a component is scheduled, it has a time associated with it. This is the time at which its next active phase is scheduled to occur. When the active phase of one component ends, it may be rescheduled (marked with the time when its next active phase will be executed), or else descheduled. In either case, the scheduled component in the system marked with the smallest time for its next active phase is resumed. Thus, the currently executed component (which may be referenced by a call on the procedure CURRENT) always has the least time associated with it; this is taken as the simulation time itself, and the simulation time jumps discretely forwards from one value to another when a new component becomes active.

In the model of the loop, three kinds of processes have been defined and have been included, together with the procedures acting on them, in different modules: the hosts, the nodes and the loop itself.

The structure of the model.

A development tool of this kind should, if possible, be independent of structure variations; these are expected to come about as the project proceeds, through technology improvements or feedbacks like those described in the introduction, point d).

Consequently, the main goals have been modularity, flexibility and programmability of the model.

Modularity has been achieved by partitioning the implementation of the simulation model into small, simple modules, which have evident meaning and functions (see fig. 1).

The model itself is logically partitioned into three "big" modules, that communicate through well-defined interfaces. The first module simulates the functions of the loop hardware, the second those of the nodes, the last those of the host set (fig. 2).

Hence it is possible, for instance, to precise or to modify in some way or another the node structure, with the sole care of not changing the interfaces with the loop and the hosts.

Interfaces definitions are built over the following interprocess communication primitives and objects:

a) SEND (Q, E, P) :

   enqueues element E into the queue Q and activates process P, if this was suspended on that queue;

b) WAIT (Q) :

   tests queue Q : if empty, the process executing the WAIT operation is suspended;

c) DELAYED-ACTIVATION (P, T, PN, F) :

   causes process P to be activated at current time + T and passes to it a buffer pointed by pointer PN; if the buffer contains useful information, the flag F is set to TRUE;

d) LIST1 objects :

   they are queues with an associated binary semaphore.

As to flexibility, the simulation model is placed on an high abstraction level, if compared with the object to be simulated: the structure of the SIMULA program modules is as simple as possible; they can "faithfully" reproduce the system only because they heavily rely on the use of parameters.

To give an example, let us consider the BASE-MODULE, about which more details

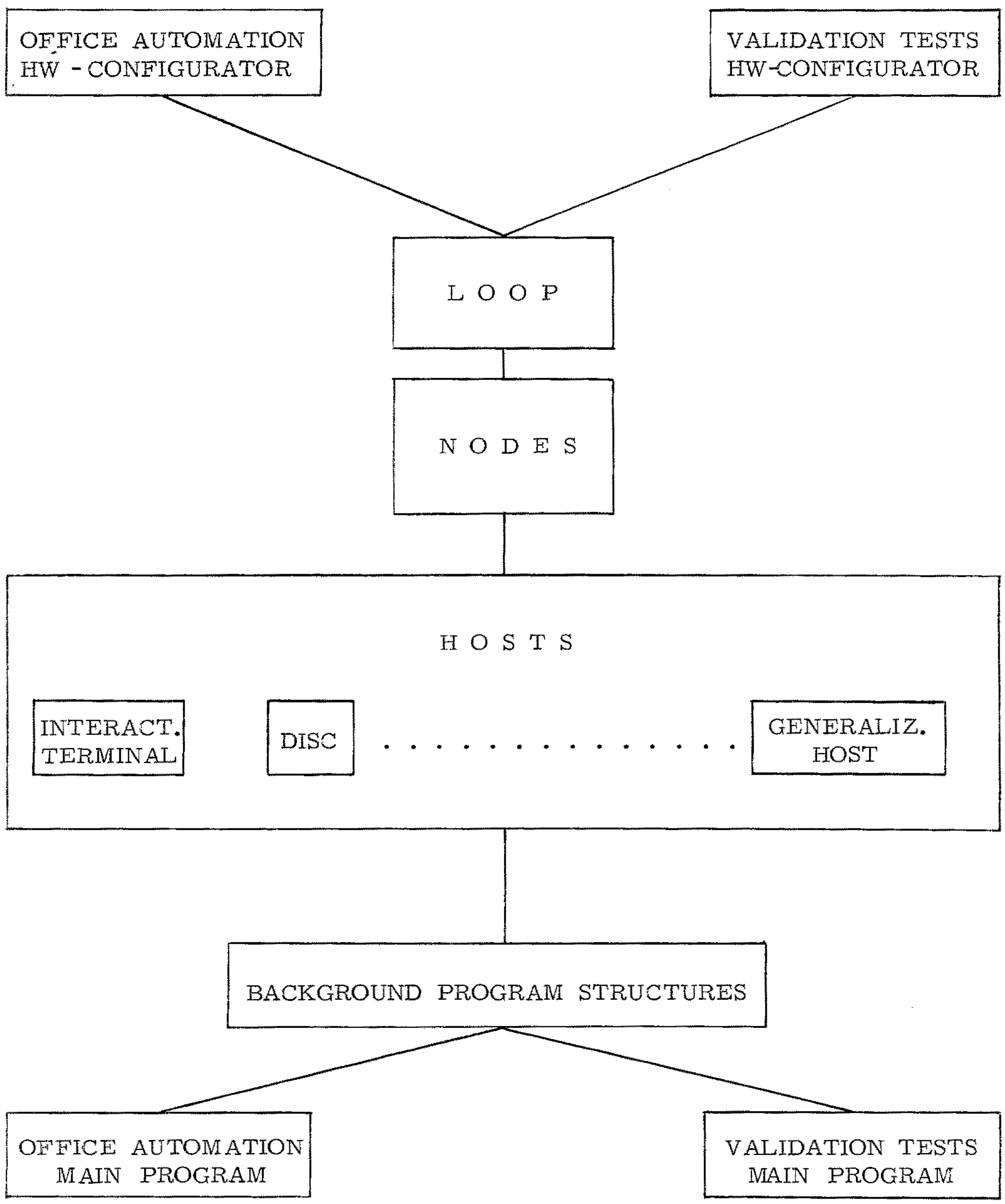
OFFICE AUTOMATION HW - CONFIGURATOR
VALIDATION TESTS HW-CONFIGURATOR
L O O P
N O D E S
H O S T S
INTERACT. TERMINAL
DISC
GENERALIZ. HOST
BACKGROUND PROGRAM STRUCTURES
OFFICE AUTOMATION MAIN PROGRAM
VALIDATION TESTS MAIN PROGRAM

Fig. 1 - Program partitioning.

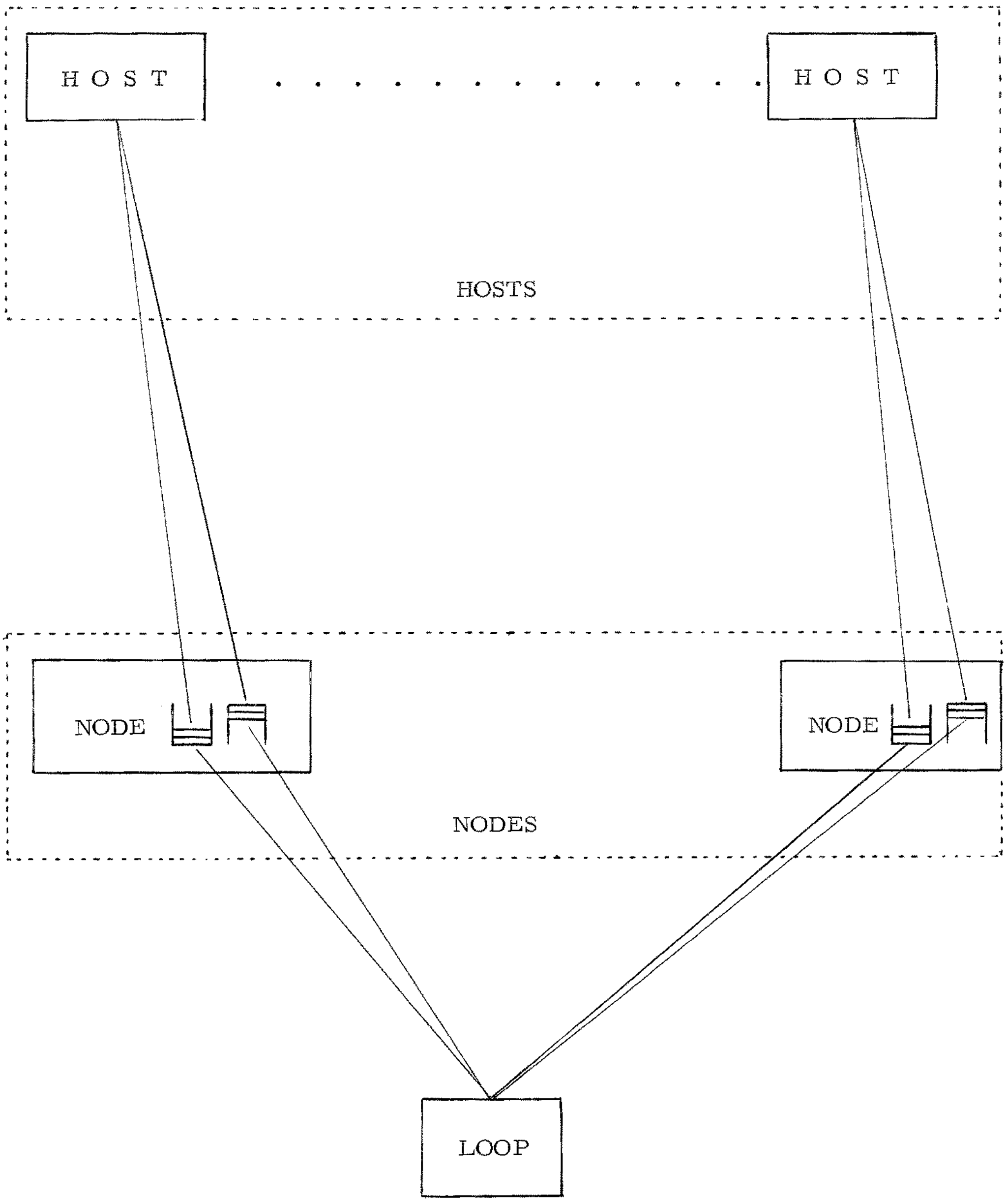

Fig. 2 - Model partitioning.

will be given in the following.

Its main part is a process called BASE-BLOCK, whose sole function is that of receiving and sending messages; in so doing it makes use of the information stored beforehand in a table. Given a chain of BASE-MODULEs, the simulation of arbitrarily complex communication protocols between them can be carried out simply by modifying, through associated procedures, the data in the tables.

All the variables meaningful for the system description have been written in the model as parameters, so as to be easily modifiable from the outside. In particular, among these there are the loop speed, the number of nodes, the "go-ahead" message length, the node delay; so it is also possible to investigate relations between these variables.

The concept of the simulation model's programmability must be understood in a particular sense. Many persons, other than those who have developed this tool, must have access to it, whenever it is to be used as a design aid for R&D groups like ours. This means that it is mandatory to hide the specific mechanisms of SIMULA, through the building of ad hoc procedures; by calling them, the SIMULA-unskilled user can tailor the model over his needs and execute the tests he wants.

But this process needs not be pushed too forward, because, anyway, the users will master programming techniques.

The procedures we have defined may be divided into two groups: those intended for the assignment of parameters to the system and for its configuration (examples 1, 2) and those intended to describe the communication protocol (example 3).

Examples:

1) NODE-NUMBER (10) : establishes that there will be 10 nodes on the loop network;
2) ASSIGN ("DISC", 3, 150, 30): the third terminal will be a disc, with mean access time of 150 msec $\pm$ 30 msec;
3) PROTOCOL ("BSC II", 4, 7, 10): the system-defined BSC II protocol will be used between the host n. 4, acting as master and terminals n. 7 - 10, working as slaves.

At present, it is not planned to implement any procedure intended to simplify for the user the definition of new protocols; yet this is quite possible, in principle. There is, instead, another possibility: a SIMULA programmer can, simply by writing a new class prefixed by BASE-BLOCK, redefine the message analysis (e.g. , ignoring the table).

The implementation of the host module.

Among the three modules, a brief description will be given of the host module, that is to say of the one intended to simulate the host set. Four classes (the class concept is a feature of the SIMULA67, see BIR73) are defined in it:
INTERACTIVE-TERMINAL, DISC, PRINTER, GENERALIZED-HOST.

Of course, different objects of these classes can be built by assigning different values to their parameters; for instance, in an INTERACTIVE-TERMINAL can be defined: time required for the transaction-initiated operations, mean time between transactions, length of the inquiry message to be sent to the discs, number and addresses (node numbers) of the discs etc.

In a DISC object can be assigned: the mean access time, the minimum and maximum length of the messages to be sent to the operators etc.

In a PRINTER object we can select the speed.

At present, the actions of these three classes are those one would expect from an operator, a disc and a printer once one has exactly defined the transactions and the background printing activity.

The tests on the simulation model have been carried out with protocols derived from the analysis of a typical office automation application. On the contrary, as the name suggests, the structure of the GENERALIZED-HOST class is neither application- nor peripheral-oriented; actually, it has been written in the hope of making easy the definition of a group of GENERALIZED-HOSTS, connected through the loop network, with different functions; and to allow the implementation of any communication protocol among them.

In this class two process types are defined (see fig. 3):

a) SW-BLOCK;

b) BASE-BLOCK.

A SW-BLOCK continuously sends commands to its associated BASE-BLOCK; it can do this either by synchronizing itself on the replies from the BASE-BLOCK, or asynchronously, with exponentially-distributed intermessage times.

The BASE-BLOCK analyzes the received commands one at a time; for any of them it decides whether, in what quantity and for whom it has to send messages through the loop. On receiving the respective replies, it repeats the same analysis, that may cause it to send other messages on the line, or to close the cycle by replying to the SW-BLOCK, and waiting for another command.

If a GENERALIZED-HOST object is to be activated only on requests coming from the loop, it must obviously be deprived of the SW-BLOCK process; the BASE-BLOCK will receive commands directly from the line.

The two processes cooperate via SEND and WAIT primitives; they share an area that contains, among other objects, two input and two output queues (for messages coming from and addressed to the loop and the SW-BLOCK); the BASE-BLOCK uses a particular table (see preceding paragraph) for obtaining all the information necessary to the message analysis.

The table, when so needed, may also specify different reply strategies for the same message, with the respective probabilities.

The configurator concept.

In the program there are some classes intended to play the role of system's configurators. It may be useful to explain with some details the function of these classes.

At present, there are an hardware configurator (the only one enrighted to assign values to system's parameters) and a software one. First of all, the configurator for a part of the system must create and link all the objects belonging to that part.

As the execution of the simulator program begins with the activation of the configurators, an automatic separation between the phases of system's generation and actual simulation is obtained.

Let us say here that the simulation model's items can be divided into two groups: "static" (intended to represent hardware pieces, processes etc.) and "dynamic" (corresponding to messages). The generation phase provides the creation of the static parts, whilst, during the simulation phase, dynamic parts (messages) are produced and acted upon by the static parts. An approach of this kind is called "machine-based".

By now, it should be clear that the definition of the configurator classes simplifies the modification not only of the values of the model's parameters, concentrated in the HW-CONFIGURATOR, but also of the system's structure: this may be accomplished, in a localized manner, by re-defining only the actions of the HW-CONFIGURATOR. To clarify this point, here a list follows of the operations involved in a GENERALIZED-HOST (a SW-CONFIGURATOR) creation and activation (refer to fig. 3):

a) the HW-CONFIGURATOR generates a GENERALIZED-HOST object, then it links this object with a node and passes to it all the parameters; among these parameters some will be needed to connect the SW-BLOCK and BASE-BLOCK processes;

b) the GENERALIZED-HOST, now acting as a SW-CONFIGURATOR, creates and links the aforesaid processes and their common data area (e.g. table for message analysis); this is local to the GENERALIZED-HOST, and becomes available to the processes because it is passed them as a common parameter by the SW-CONFIGURATOR;

c) at this point, the HW-CONFIGURATOR compiles, inspecting the host data area, the table entries, thus defining the communication protocol for that host.

This operation is accomplished through the execution of specific procedures local to the HW-CONFIGURATOR.

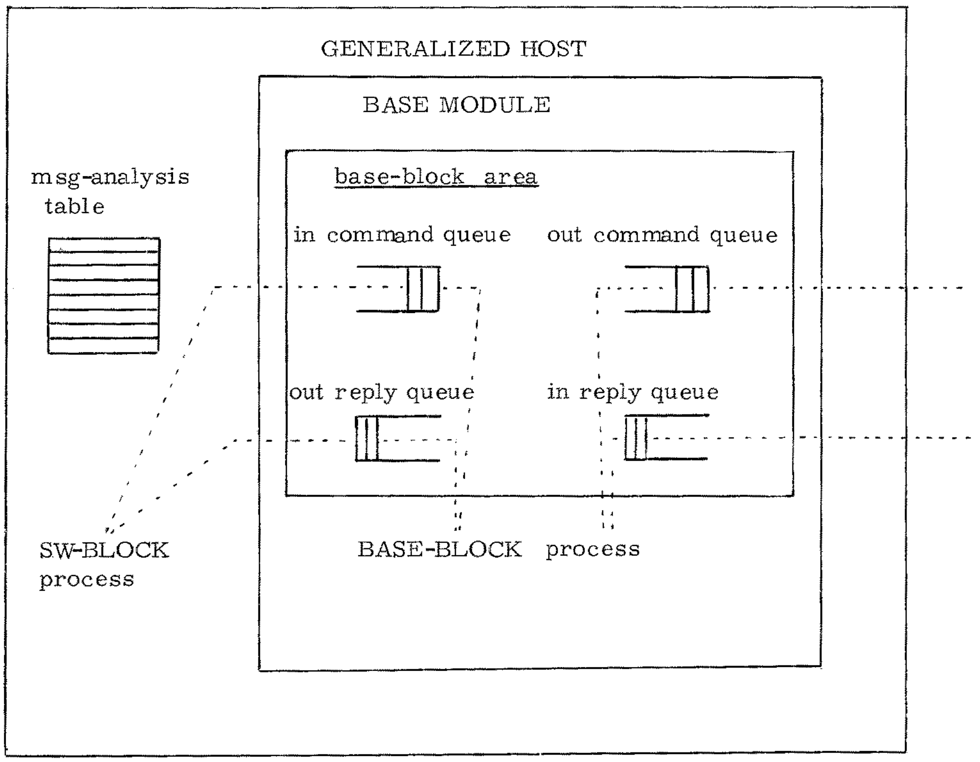

Fig. 3 - Generalized host's structure.

RESULTS.

First of all, the model has been used to obtain the numbers that have been required in order to achieve the primary goal: the preliminary definition of the system for a typical office automation utilization. In the following pages some of the more typical curves (fig. 4, 5, 6) are given.

A simple mathematical model has been written for the loop network; it relates some of the more meaningful variables : transmission speed, node delay and number, utilization of the loop; this analysis and the simulation results both point to the utilization of the loop as a faithful measure of the system's level of crowding.

The utilization of the loop is defined as:

$$O = \frac{\text{MESSAGE TRANSMISSION TIME}}{\text{TOTAL TIME}}$$

The curves in fig. 3 give a visual confirmation of the preceding statement: they refer to a simple, though unrealistic, situation in which the load on the network (number of transactions per minute, of lines to be printed etc.) varies with time, reaching only a minimum or a maximum value.

The broken line shows the load variations: in the second curve, every "X" represents a sample of the loop's utilization and the resulting curve follows the preceding one with a short delay: from this one can have an estimate of the response time of the system.

In the condition of the maximum load, the system is at its stability limit: correspondingly, we can see that the third curve (sampling the number of elements in the mostly used transmission queue) oscillates strongly. The oscillations are caused by a sort of "noise" in the input to the system (e.g. the arrival times for the transactions are randomly varied with a Poisson distribution; only the mean interarrival time can be assigned); so there is a good probability that, at a certain moment, the fluctuations in the message production rates of the various nodes sum up to cause an overcrowding, with a resulting increase in the number of messages waiting to be transmitted. This can be seen as a "micro-breakdown" of the system .

Shortly afterwards, usually, the equilibrium returns, but it takes a certain time to reduce the number of elements in the queues, unless an opposite fluctuation occurs.

At present, it has not yet been possible to compare these curves with the actual behaviour of a real system, since the implementation of the loop is still in progress.

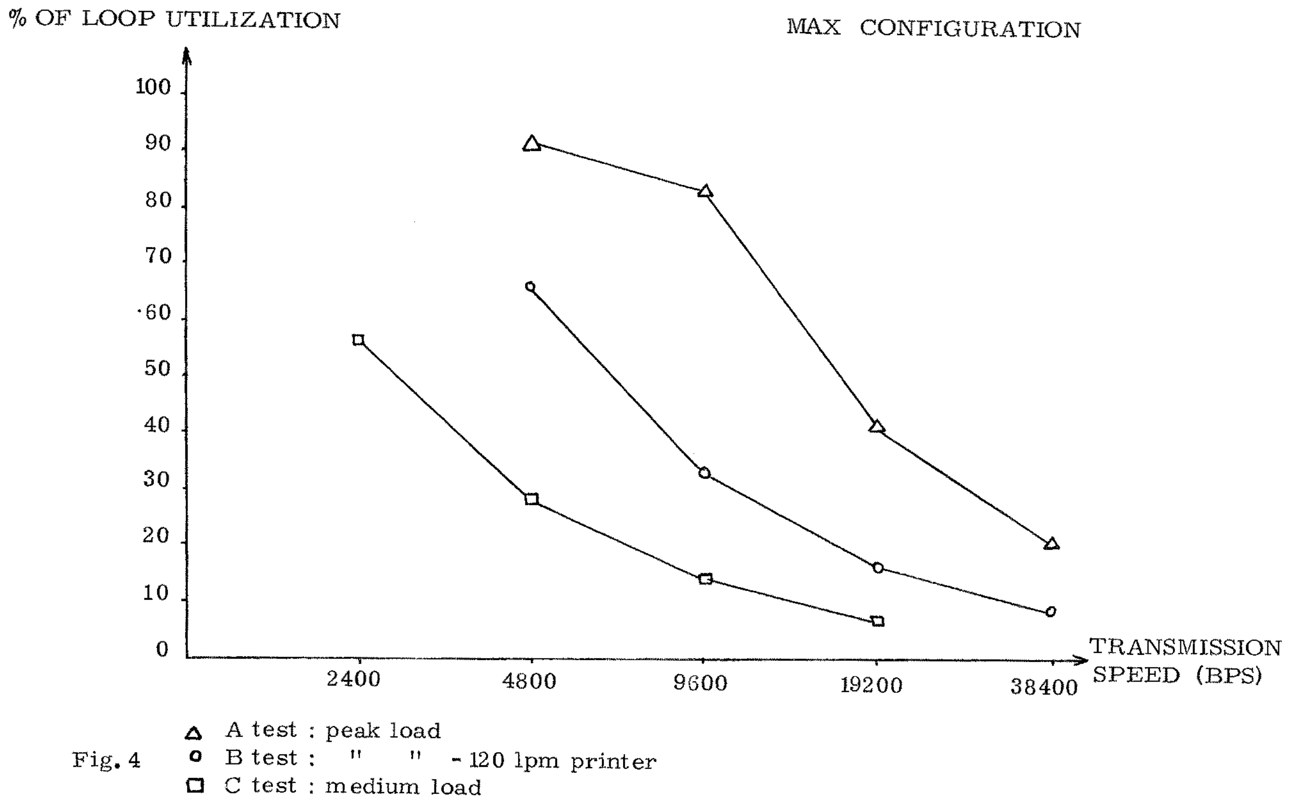
% OF LOOP UTILIZATION
MAX CONFIGURATION
100
90
80
70
·60
50
40
30
20
10
0
2400
4800
9600
19200
38400
TRANSMISSION SPEED (BPS)
△ A test : peak load
○ B test : " " - 120 lpm printer
□ C test : medium load

Fig. 4

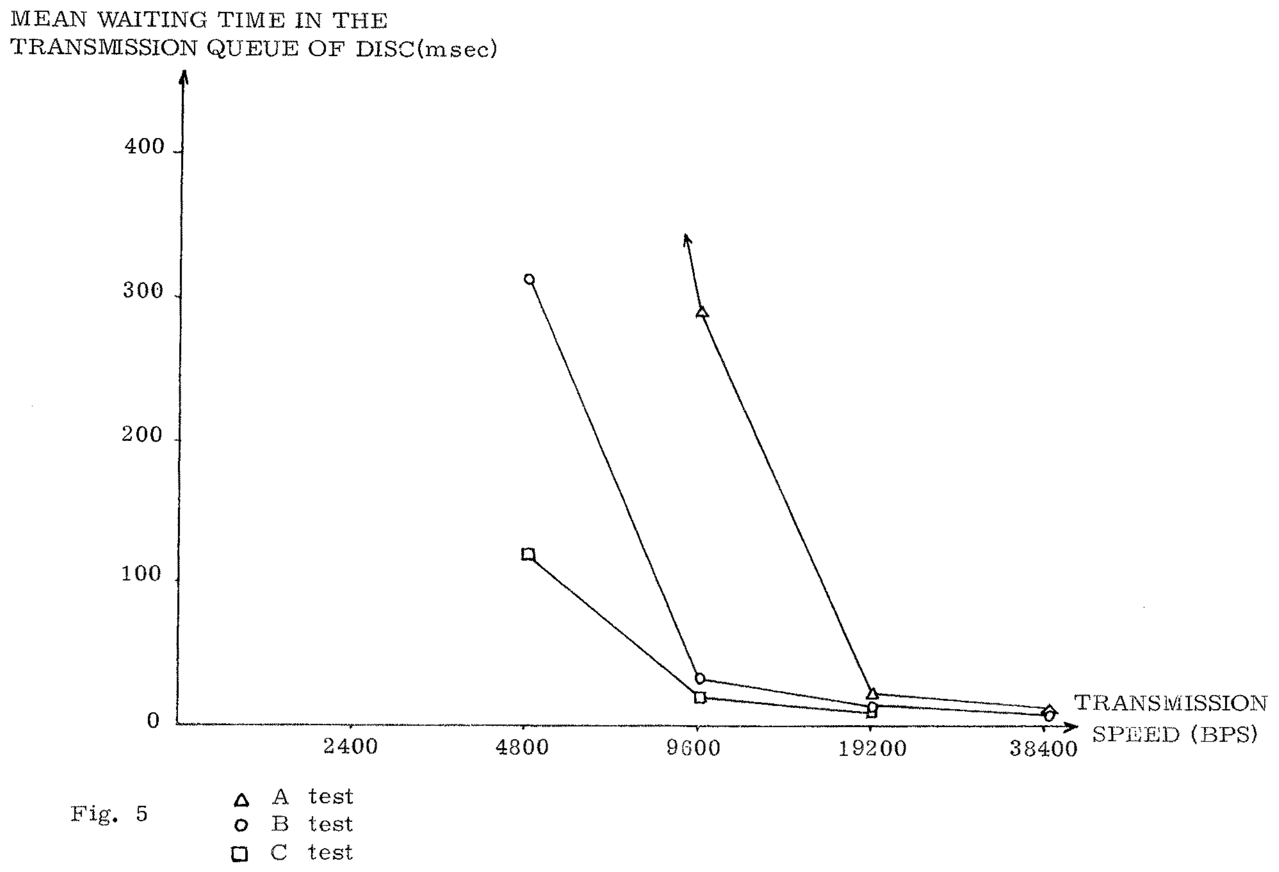
MEAN WAITING TIME IN THE
TRANSMISSION QUEUE OF DISC(msec)
400
300
200
100
0
2400
4800
9600
19200
38400
TRANSMISSION
SPEED (BPS)
△ A test
○ B test
□ C test

Fig. 5

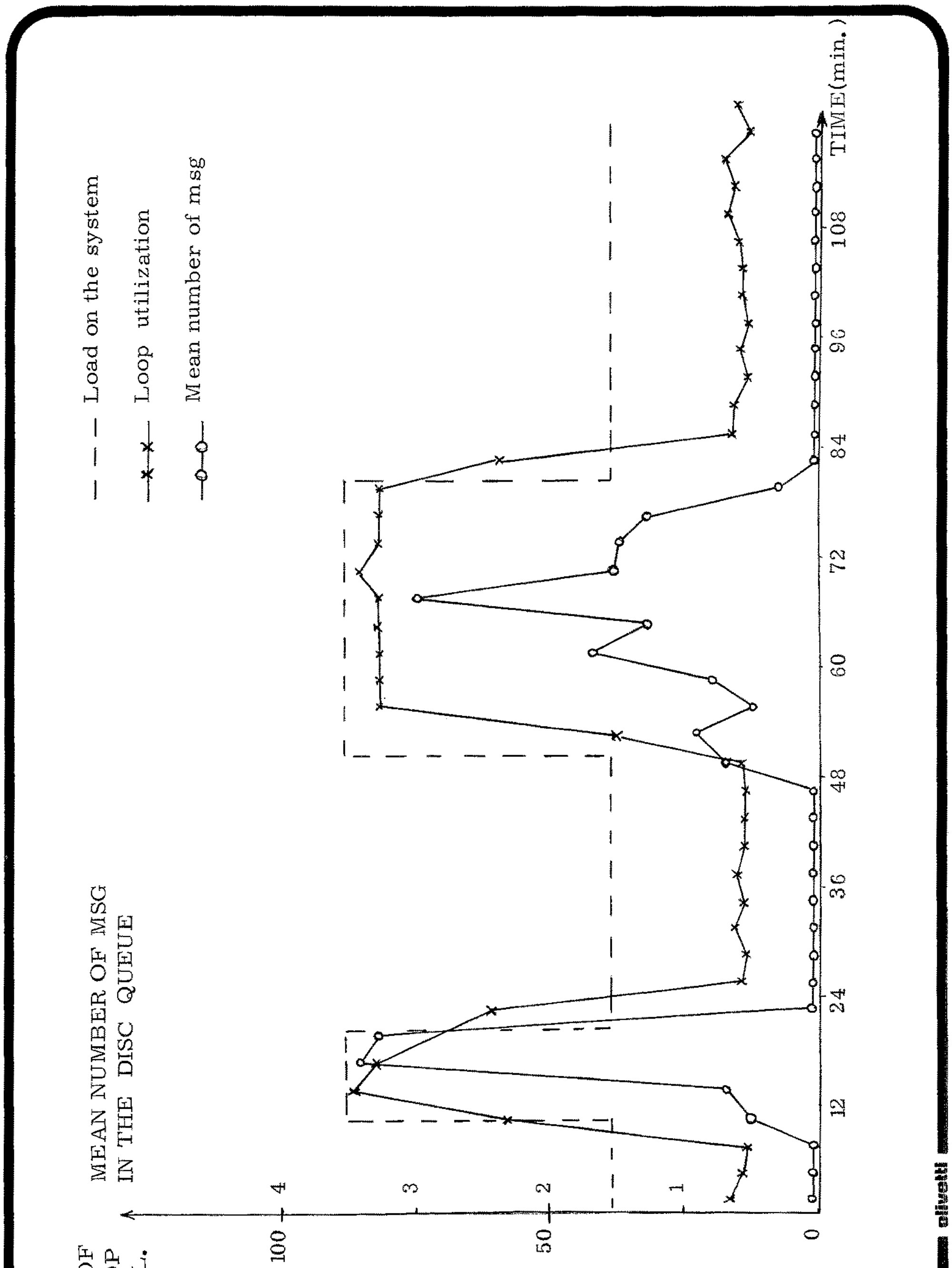
% OF LOOP UTIL.
100
50
0
MEAN NUMBER OF MSG IN THE DISC QUEUE
4
3
2
1
12
24
36
48
60
72
84
96
108
TIME(min.)
Load on the system
Loop utilization
Mean number of msg
olivetti

Fig. 6

CONCLUSIONS.

The proposed simulation model has already achieved some of its goals: its flexibility has allowed us to obtain quickly the required curves and to show the system's behaviour with quite different configurations, transmission speeds, node delay etc.

At present, the work on the model proceeds in the direction of software simulation and of the study of other possible applications of the architecture.

From the first results, some ideas have been suggested; as it has been shown in the preceding paragraph, the utilization of the loop is a good measure of the system's level of crowding. In a real system, it is desirable to avoid the micro-breakdowns; then one can imagine a prevention technique based on the estimate of the loop's utilization, easily made by every node. When this measure signals that the system has reached a nearly critical state, the nodes with less urgent messages or with a small number of elements in their transmission queues can decide to temporarily slow down their transmission rates.

The approximate length of the various modules of the simulation program are given: loop and nodes modules, 80 code lines each; host module, 370 lines, of which 100 have been written for the GENERALIZED-HOST. The total length of our program is 800 lines, not including the procedures written to simplify the model's programming. These data show that the proposed simulation approach is simple to implement, and that the model's modularity and flexibility are obtained with a modest program length.

The simulation program has been run on an IBM 370/168, as we have already remarked; the required CPU time of this computer is, of course, a function of simulated time but also of the memory used, the kind of tests etc. The mean ratio between simulated time and 370 CPU time is approximately 40.

BIBLIOGRAPHY.

BIR73 : Birtwistle, Dahl, Myhrhaug, Nygaard - SIMULA BEGIN - Petrocelli Ch. 1973.

CAS76 : G. Casaglia, S. Copelli, N. Lijtmaer - DISTRIBUTED CONTROL FOR LOCAL NETWORK OF MINIS : A DESIGN APPROACH USING MICROPROCESSORS - Euromicro Symp. (p. 119), oct. 1976.

FAR72 : D. J. Farber, K. Larson - THE STRUCTURE OF A DISTRIBUTED COMPUTER SYSTEM - THE COMMUNICATION SYSTEM - Proc. Symp. on Comp. Comm. Network and Teletraffic, Pol. Inst. of Brooklin Press, apr. 1972.

FRA74 : A. G. Fraser - LOOPS FOR DATA COMMUNICATIONS - Bell Lab., Comp. Science Tecn. Report n. 24, dec. 1974.

GOR69 : G. Gordon - SYSTEM SIMULATION - Prentice-Hall, 1969.

KLE75 : L. Kleinrock - QUEUING SYSTEMS - Wiley and sons, 1975.

NEW69 : E. E. Newhall, W. D. Farmer - AN EXPERIMENTAL DISTRIBUTED SWITCHING SYSTEM TO HANDLE BURSTY COMPUTER TRAFFIC - Proc. ACM Symp., Pine Mountain, Georgia, oct. 1969.

PIE72 : J. R. Pierce - NETWORK FOR BLOCK SWITCHING OF DATA - Bell Syst. Techn. Journal, jul.-aug. 1972.

REA76 : Reames, M. T. Liu - DESIGN AND SIMULATION OF THE DISTRIBUTED LOOP COMPUTER NETWORK (DLCN) - Proc. 3rd Ann. Symp. on Comp. Architecture, jan. 1976.

ZIE75 : B. P. Ziegler - THEORY OF MODELLING AND SIMULATION - Wiley and sons, 1975.

ACKNOWLEDGEMENTS.

We wish to thank N. Lijtmaer, of the Istituto per l'Elaborazione dell'Informazione, Pisa, and ing. F. Tisato, Politecnico di Milano, for their suggestions on the best approach to the problem and ing S. Brandi, Olivetti, for having helped us in the definition of the program structure.