COMMUNICATION AND SYNCHRONIZATION TOOLS IN A DISTRIBUTED ENVIRONMENT*

H. Le Goff, G. Le Lann

IRISA, Université de Rennes, BP 25A

35000 Rennes, France

Key words : computer networks, distributed systems, communication protocols, flow control, resource utilization.

Characterization of a distributed environment

Most of computer systems are operated and controlled from a central unit which is in charge of running an Operating System. This kind of architecture is not flawless : it is not easily expandable, not very efficient because of the potential bottleneck at the central unit level and is highly vulnerable.

Multiprocessors were first designed to speed up computation, by taking advantage of the inherent parallelism of these machines and to increase their reliability. We see now that they have the potentiality of being operated in a decentralized mode. Nevertheless, in most cases, some kind of centralization can be found inside these multiprocessors : one processor is monitoring the others, internal communications are possible only through the common memory space, all the tasks to be run are queued up in a unique location, and so on.

* This work was supported by the Direction des Recherches et Moyens d'Essais, Ministère de la Défense, contrat n° 74/540.

Up to now, the best examples of real distributed systems are computer networks [1], [2], [3]. Computer networks behave according to some specific sets of rules called Protocols which relieve network processors from the need for a central "controller" in order to perform their own tasks. As will be shown later, processors having equal rights and equal responsabilities are said to belong to the same functional layer and this decomposition leads to a hierarchical architecture.

We think that centralized and distributed systems differ from each other on two important points :

- entities willing to communicate do so directly in a distributed system, instead of referring first to a unique monitoring process which is then responsible for establishing communications.
- it is meaningless to define the "state" of a distributed system.

The communication and synchronization tools to be described in this paper are intended for general purpose computer networks and are not tied to a specific system. These tools can thus be used in any distributed system, whatever its size.

Servers in a distributed system. The transparency concept

Locating a process in a distributed system like a computer network is a many solutions problem, each of these solutions having some drawbacks. For instance, processes can be accessed by the means of physical addresses ; in that case, users have to be aware of the existence of all processes (updating problems) and are bound to the naming convention of the hosts where these processes reside ; processes are not allowed to migrate on different hosts and their availability is thus directly dependant upon the host's failures ; furthermore, fair load sharing is very hard to achieve.

Another solution is to access processes by using logical names. One then must be able either to handle directly logical names or to map logical names into physical addresses ; the first solution has been chosen for the DCS network [4] where logical names are recorded into associative memories distributed over a ring topology allowing for natural broadcasting ; the second solution is used in the Cyclades subnetwork [5] in which a Transport Station name is translated into a physical node address by accessing a table.

In that latter case, the number of processes has to be kept rather small in order to avoid large search times. Consequently, this approach does not seem to be workable with private user processes because of their dynamically evolving nature. On the contrary, this solution is quite appropriate for standard processes like system Loggers, Compilers or library Subroutines which we call Servers.

Assuming now that many different Servers perform a specific service, for instance NLED(*) Compilation, we are faced with the problem of selecting one of the Servers when receiving a corresponding request. The user issuing the request has to be kept unaware of that situation for many reasons ; first of all, he is surely not interested in that problem because he is currently trying to solve another one ; second, if we want the user to be provided with a good service, we have to achieve automatic load sharing and the user is obviously not the best person to perform that internal balancing ; third, Servers failures should be ignored as far as the user is concerned and this is also directly related to load sharing.

This is what we call transparency and some methos are currently being investigated which allow for a decentralized allocation of resources [7], [8] and internal load sharing into distributed systems [6] ; a diffusion technique, similar to the Arpanet adaptive routing mechanism has been proposed for distributed network topologies ; another scheme, the circulating vector technique, is based on the virtual ring concept and is usable on any physical topology.

With these methods, automatic selection of one server among many is possible ; the current physical location of that server is then notified to the user and from this point, communication and synchronization between the user and the server proceed on a conventional point to point basis.

In what follows it will be shown how communication and synchronization are performed for a specific class of processes in a computer network. Evaluation of a very general flow control mechanism has been undertaken ; a conflicting optimization case is then reported which is very similar to some situations observed in hierarchical and centralized Systems.

(*)Nicest Language Ever Designed, precisely the Language you are in love with.

<u>Interprocess communication and synchronization in a computer network</u>

In conventional systems, processes willing to exchange data do so according to pre-defined rules. Most of the time, implicit assumptions are made ; for instance, the data exchange is error free. In computer networks, stronger constraints have to be dealt with : data may be corrupted during the transfer, time references are not identical for the sender and the receiver, a common physical space is not shared by the sender and the receiver, data transmitted are not kept is sequence.

Consequently, a good transmission tool is such that :
- no desynchronization may occur between the communicating processes
- transmission errors are rapidly detected and corrected
- flow control is performed efficiently.

As a result, for the processes, the transmission delay will be minimum and the available throughput will be maximum [9].

Communication networks handle standard data blocks called packets. Packets are size-limited for many reasons and the maximum size value depends on the transmission medium characteristics.

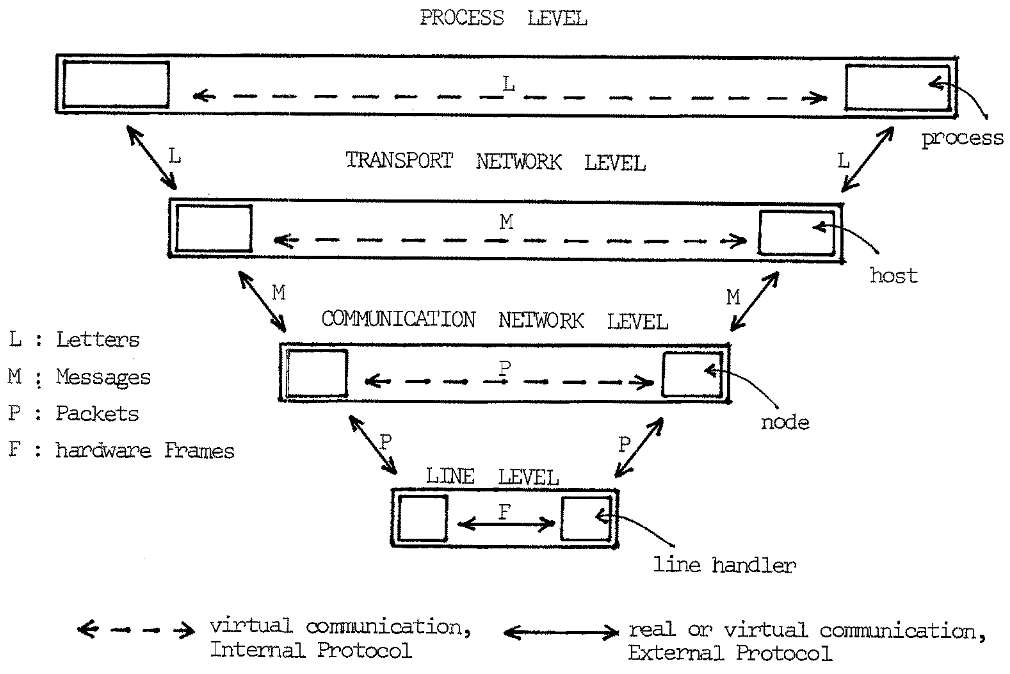


FIGURE 1

User processes exchange data records called Letters ; Letters which do not fit into one packet have to be fragmented before being transmitted and need to be reassembled before delivery ; a fragment of a Letter is called a Message (see figure 1).

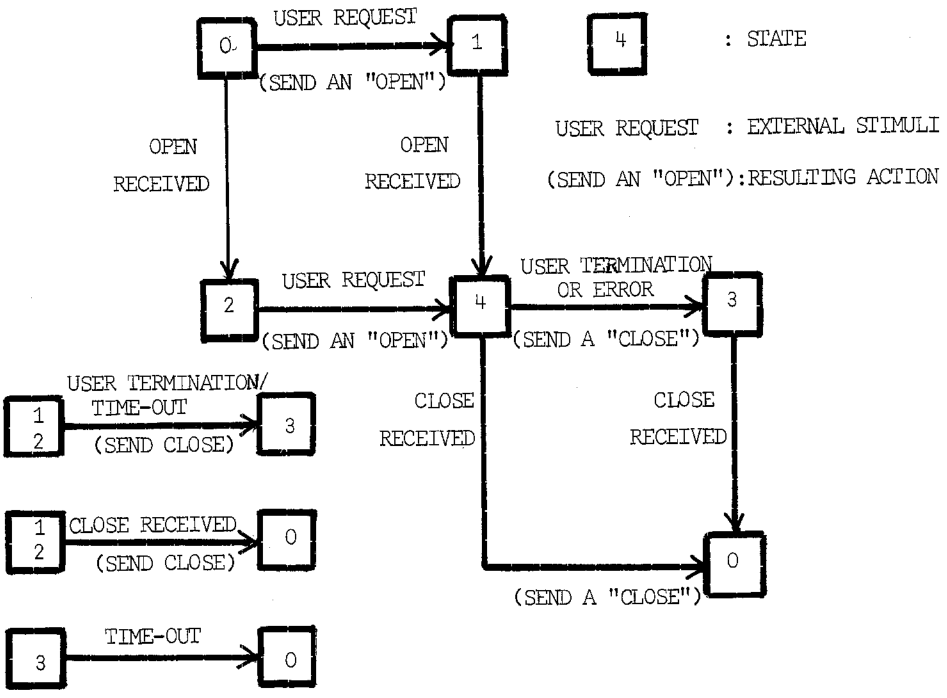
Processes are run on physical hosts and several processes may initiate simultaneous communications from one host ; moreover, any given process should be allowed to handle many concurrent communications too. Then the need for a multiplexing/demultiplexing function and for an activation/termination function.

How can transmission errors be rapidly detected and corrected ? Each packet carries some extra-information allowing for error detection (CRC, for instance). The receiving process is then able to ask for a transmission if needed. But this does not work if packets are lost ; another scheme has to be devised. Most of communication Protocols use a Timer + Positive Acknowledgment mechanism ; for each packet, if a preset time interval has elapsed before receiving the corresponding acknowledgment then a retransmission of this packet takes place. Obviously if timeout values are too short or packet transmission delays are sometimes too large, duplicates of packets will be created. These duplicates must be detected. This is an easy task if only one packet at a time is travelling between two processes ; an alternate numbering scheme may be used in that case. For the sake of efficiency, some "anticipation" should be allowed and many packets may be outstanding on the virtual path existing between two processes ; such a path is called a Liaison. Unique identification of Letters on a Liaison is then achieved by the means of a cyclic numbering scheme, the cycle value being such that no confusion may occur.

One upper limit for the number of outstanding packets is given by the cycle value. Another one is given by the amount of buffers allocated by the receiving process. Flow control purpose is to monitor this parameter dynamically and to adjust the output rate of the sending process to the variable input rate of the receiving process.

All these functions are standard requirements as far as processes are concerned and should be made available on any given host. The corresponding software is called a Transport Station (TS). Processes willing to communicate with each other do so by accessing their local TS. Control of actual transmission is performed by the means of a Transport Protocol.

Simulations performed for the Cyclades project [10] showed that desynchronizations cannot be totally avoided when opening or closing a Liaison ; it was shown also that absolute credit values should not be used in flow control ; credit values are said absolute when not refering to a specific location in the data flow. An example of desynchronisation is given on figure 2. This uncertainty requires a layered approach in which desynchronizations at a given level can be detected and


USER REQUEST
(SEND AN "OPEN")
: STATE
USER REQUEST : EXTERNAL STIMULI
(SEND AN "OPEN"):RESULTING ACTION
OPEN RECEIVED
OPEN RECEIVED
USER REQUEST
(SEND AN "OPEN")
USER TERMINATION OR ERROR
(SEND A "CLOSE")
USER TERMINATION/ TIME-OUT
(SEND CLOSE)
CLOSE RECEIVED
CLOSE RECEIVED
CLOSE RECEIVED
(SEND CLOSE)
(SEND A "CLOSE")
TIME-OUT

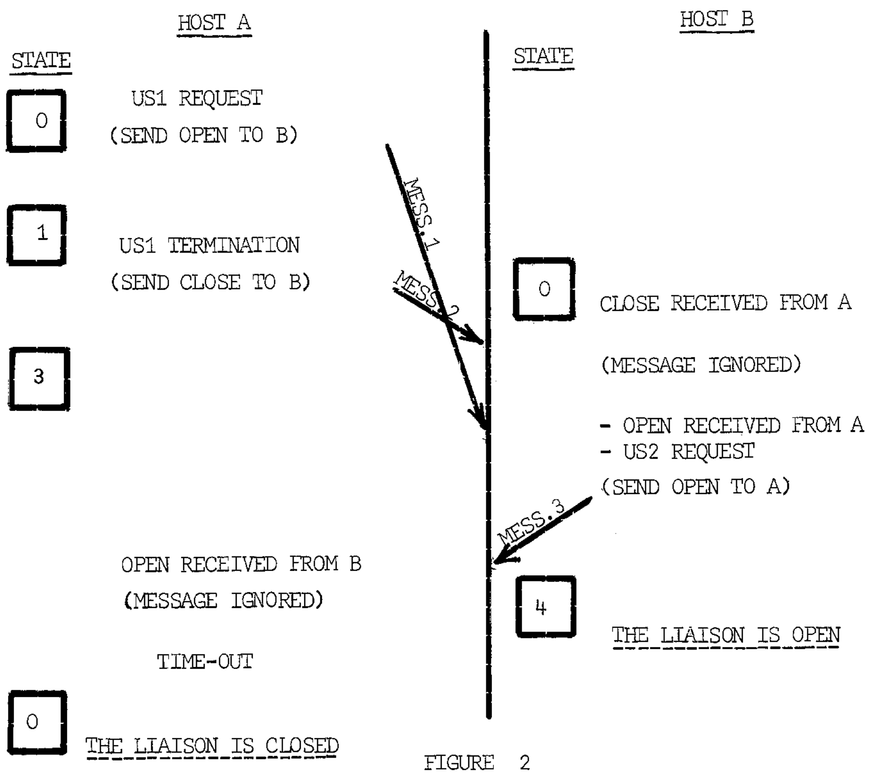
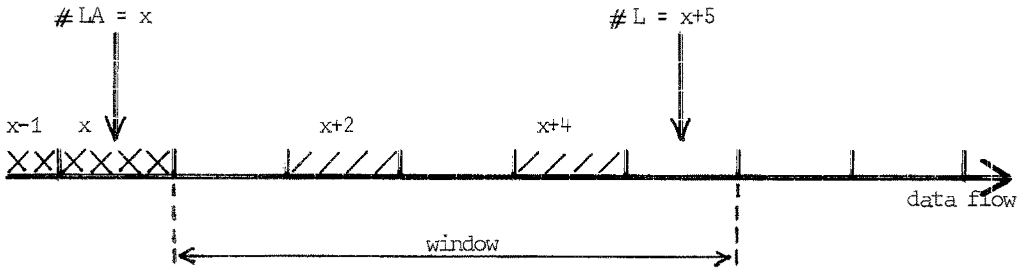

HOST A
HOST B
STATE
STATE
US1 REQUEST
(SEND OPEN TO B)
MESS.1
US1 TERMINATION
(SEND CLOSE TO B)
MESS.2
CLOSE RECEIVED FROM A
(MESSAGE IGNORED)
- OPEN RECEIVED FROM A
- US2 REQUEST
(SEND OPEN TO A)
MESS.3
OPEN RECEIVED FROM B
(MESSAGE IGNORED)
TIME-OUT
THE LIAISON IS OPEN
THE LIAISON IS CLOSED

FIGURE 2

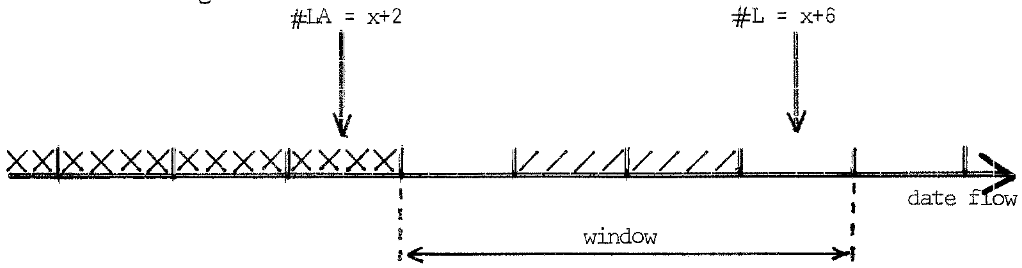


buffer space at the receiving TS at time T0
(Messages x+1 and x+3 missing)

at time T1>T0, the receiving TS sends Message (#LA, 5) to the sending TS

between T1 and T2>T1 : - reception of letters/fragments x+1 and x+5
- the user provides a new buffer

time at the
receiving TS



Legend :

XXX received and acknowledged

/// received but not acknowledged

#LA : last acknowledged reference

#L : upper limit allowed in reception for the receiver and in transmission for the sender

Copies of unacknowledged Messages are kept by the sender for a possible retransmission.

FIGURE 3

corrected at higher levels.

One widely accepted flow control scheme is the Window mechanism, first introduced in Cyclades. With such a scheme, Messages sent on to a Liaison are given sequential references ; because of transmission failures or race conditions, resequencing of Messages has to be performed by the receiving TS which acknowledges the data flow up to the first missing Message (see figure 3) ; credits are indicated as incremental values refering to the last acknowledged reference ; acknowledgments Messages are periodically issued ; loss or duplication of these Messages do not lead to desynchronization.

In order to get a better insight into the problem of designing interprocess communication Protocols, description and evaluation of one INWG* proposal is now reported. This Protocol, the ZE Protocol [11] makes use of the Timer + Repetition, Positive Acknowledgment and Window technique :

- error control
When fragmentation is on (Letters not fitting into one packet), the ZE Protocol requires one acknowledgment per whole Letter only. When a Letter is timed out, the ZE Protocol provides for the retransmission of that Letter and all subsequent Letters.
- flow control
The ZE Protocol requires an agreement to be reached by the processes at the Liaison set-up time, on a common Letter size ; then, the credit values are indicated in Letters.

Evaluation results (figure 4)

The user throughput versus credit is a step function. If n is the Letter length, the same throughput is achieved for all credit values included in the interval [Kn, (K + 1) n[, K integer. This is due to the flow control policy requiring an allocation of a buffer of size n before a transmission of a Letter takes place. Thus, to achieve a given throughput, more space is needed as Letter size increase ; for two different sizes n and m with n < m, the ratio of the credit values required in both cases to achieve the same throughput is smaller than m/n.

For small credit values, for instance 700 octets, it is more efficient to transmit short Letters (120 octets) instead of long Letters (480 octets). We felt necessary to investigate this phenomenon in more depth.

* International Network Working Group, IFIP WG 6.1

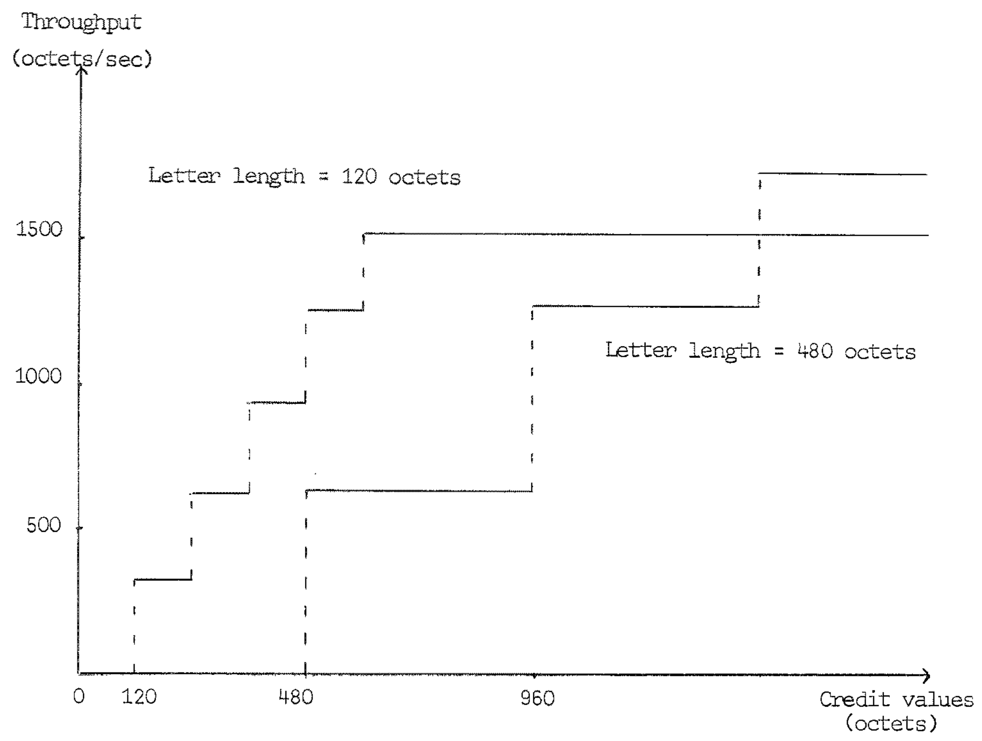


Figure 4

<u>Optimal resource utilization</u>

Both at the communication network level and at the line level, the important goal to achieve is a good utilization of the offered transmission facility. Most of Network designers have agreed on a basic packet size ranging around 2000 bits. The fixed overhead payed per packet transmission seems to be quite acceptable for such a size.

But what is the real overhead to be experienced when the full packet capacity is not used ? This happens for instance with interactive traffic. Most of interactive letters are short - a few characters - and obviously, a variable fraction of the potential throughput is wasted in that case. It is usually accepted that wasting the throughput is the price to pay in order to achieve fast transmission, which is precisely the important criteria in interactive applications. But what delay and what throughput ? If it is true to say that a short transmission time is important to the user then, the right question to ask is how much of the throughput is wasted at the user level too and not at the communication network level. Optimization of that latter level is another problem and global optimization of both the communication and the transport levels may turn out to be a tradeoff.

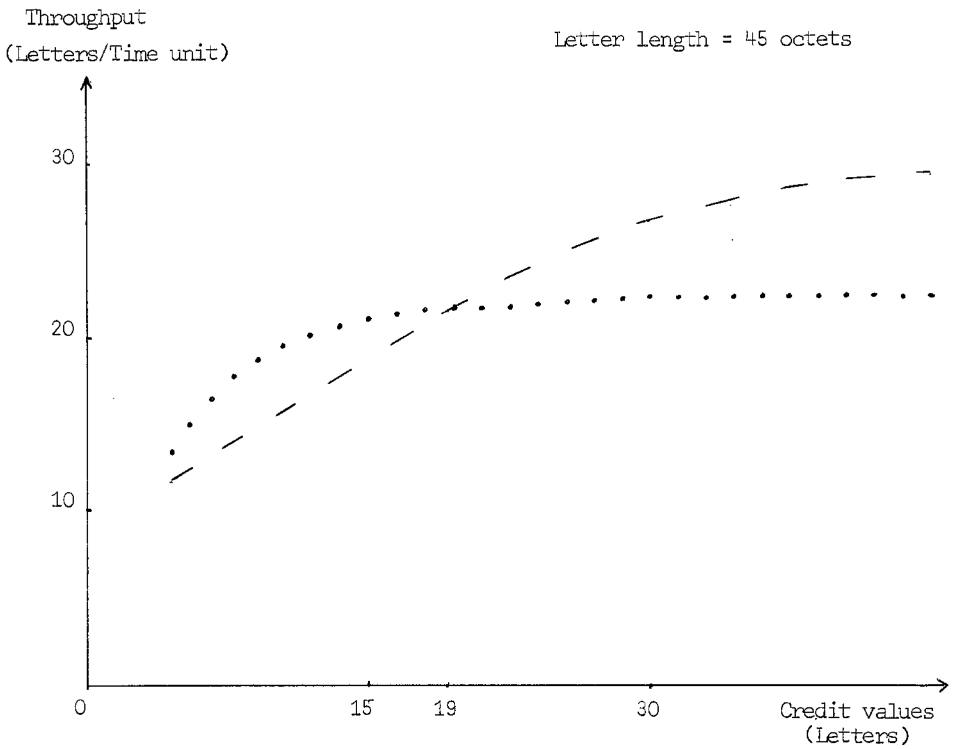


Figure 5

User Letters are handled by Transport Stations which are in charge of both error control and flow control. Then, the highest achievable throughput is the one allowed by the TS flow control mechanism.

Results of a simulation study are shown in figure 5 (several cases have been simulated). Multiplexing several short Letters into packets has been tried against the one Letter per packet case. These results indicate very clearly the existence of a threshold value, T. Multiplexing is efficient only when credit values larger than T are made available by the receiving TS ; for smaller credit values, optimal user delay and throughput are achieved by not multiplexing Letters into packets. This is due to the fact that multiplexing letters into packets leads to longer packets which are much slower than the one letter packets to travel across the various network layers ; for the same reason, acknowledgments being transmitted with the reverse traffic are slower to come back ; new credit values are carried with these acknowledgments ; it may happen then that the sending TS having spent all its credits has to wait for a future acknowledgment carrying a non-zero credit value before resuming the letter transmission. This leads to a non-optimal throughput.

Interestingly enough, the most realistic values for a credit on a Liaison are values smaller than T ; this is even more obvious as several Liaisons may be in use simultaneously between two Transport Stations.

Thus, good utilization of a layer in a computer network may require some specific tools which are conflicting with the optimization of the adjacent layer. This is a well known situation in conventional structured Systems ; the same problem arises in computer networks and distributed systems and some further works are needed in order to determine what the real tradeoffs are.

Conclusion

In this paper, synchronization and communication tools designed to operate in a distributed environment have been described. Good insight into their behaviour has been made possible through simulation studies. Future Operating Systems may have to be run on distributed architectures and may need to include that kind of mechanisms. Moreover, another class of problems has been reported which deal with the internal control and the non-centralized allocation of resources into such distributed systems. The transparency concept and some others currently being investigated may help to build efficient distributed architectures.

References

[1] BARBER D., The European computer network project, Washington, ICCC 1972, pp. 192-200.

[2] ROBERTS L., WESSLER B., Computer network development to achieve resource sharing, SJCC 1970, pp. 543-549.

[3] POUZIN L., Presentation and major design aspects of the Cyclades computer network, 3rd Data Communication Symposium, Tampa 1973, pp. 80-87.

[4] FARBER D. et al, The Distributed Computing System, 7th Annual IEEE Computer Society International Conference, 1973.

[5] POUZIN L., Cigale, the packet-switching machine of the Cyclades computer network, IFIP 1974, pp. 155-159.

[6] LE LANN G., Une approche des futurs réseaux informatiques, Congrès International sur les Mini-ordinateurs et la Transmission de Données, Liège 1975.

[7] LE LANN G., NEGARET R., Operating principles for a distributed multimicroprocessor, First Euromicro Symposium, Nice 1975, pp. 219-222.

[8] LEHON A., NEGARET R. and LE LANN G., Distribution of access and data in Large Data Bases, International Symposium on Technology for Selective Dissemination of Information, Rep. di San Marino, 1976.

[9] LE GOFF H., PEDRONO R., Les Protocoles de Transport dans les réseaux à commutation par paquets : présentation et évaluation, International Telecommunication Union, Genève 1975 Symposium, pp. 3.5.6.1.-3.5.6.9.

[10] LE LANN G., La simulation et le projet Cyclades, Congrès Afcet Informatique et Télécommunications, Rennes, 1973, pp. 297-304.

[11] ZIMMERMANN H., The Cyclades end-to-end protocol, 4th Data Communication Symposium, Québec City, 1975, pp. 7.21-7.26.