CERN/EAST note 94-22
July 28th 1994
draft 2

Communication benchmarks with the Ancor Fibre Channel Fabric

Fabrice Chantemargue

I. Introduction

The aim of this work is to measure communication times for transferring data from memory to memory between *RS/6000* IBM workstations via a 16 port Ancor Fibre Channel Fabric (or Switch). In fact, not only the performance of the switch is measured, but also the performance of the interfaces and the software (drivers).
Figure *1* shows the experimental hardware configuration used. One machine is a *250* model with a *PowerPC 601* processor at 66 MHz, the second one is a *590* model with a *Power2* processor at 66 MHz, and the third one is a *C10* model with a *PowerPC 601* at 80 MHz. A *MCA CIM* (Micro Channel Architecture Communications Interface Module) *250* Ancor adapter is located in each workstation to provide access to the Fibre Channel world

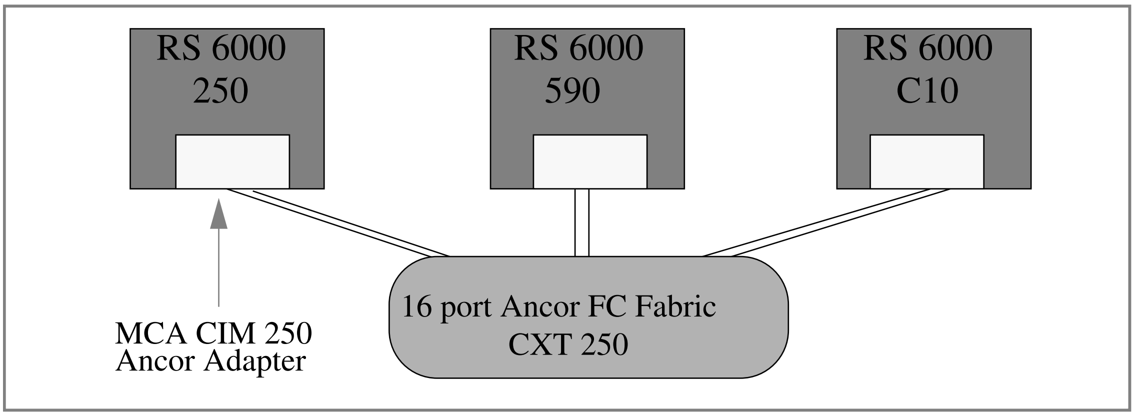


Figure 1: Experimental hardware configuration.

The *MCA CIM 250* interface to the *MCA* bus (*RS/6000* IBM workstation) is a 32 bit bus Master capable of data transfer rates up to 40 MBytes/s and it should not represent a bottleneck in this system, where the speed of each 50 micron optic fibre is 266 Mbits/s. For more details about the Fibre Channel Standard, the Fibre Channel fabric and adapters, see [1], [2], [3] and [4].
The driver available for *MCA CIM 250* adapters can support two different interfaces:

- *TCP/IP* interface,
- Direct Channel interface [4].

Much of the overhead of *TCP/IP*'s reliability checks is redundant, because the *FCS/MCA* system protects the integrity of data transfers. The Direct Channel interface feature of the driver allows the replacement of the *TCP* and *IP* levels of protocol with a direct connection to the driver, thereby achieving the highest possible bandwidth.

We performed transmission tests using both interfaces albeit for our application we are interested in using the most optimized one (providing the best bandwidth), viz the Direct Channel interface. Results are reported hereinafter.

II. Results using *TCP/IP*

The program used to test out the performance is 'ttcp'. This program is in the public domain and was originally created by the US Army Ballistics Research Lab (BRL).
There are three input parameters to this program, which can be chosen independently:

- the size of the message to be sent,
- the number of times the message is sent (to perform an average measurement),
- the socket buffer size.

The size of the message to be transferred ranges from 1 byte up to 8000 bytes. There is no limitation on the message size at the user level; we have chosen this interval because it is representative of the application in which we are involved.
The number of times the message is sent is equal to 10^5 if the message size is lower than 512 bytes. In the case of a message size greater than 512 bytes, the number of times the message is sent is equal to 10^4. These values have been chosen like this in order to provide relevant time measurements.
Seven values (in bytes) for the socket buffer size have been tested: 1K, 2K, 4K, 8K, 16K, 32K and 64K (1K is equal to 1024 bytes)
Graphs (see figures *2a* and *2b*) show how the communication time and therefore the communication speed scale with the size of the message to be transferred and the socket buffer size. Figures *2a* and *2b* display results when the *590* workstation acts as a sender and the *250* workstation acts as a receiver, and vice versa respectively.

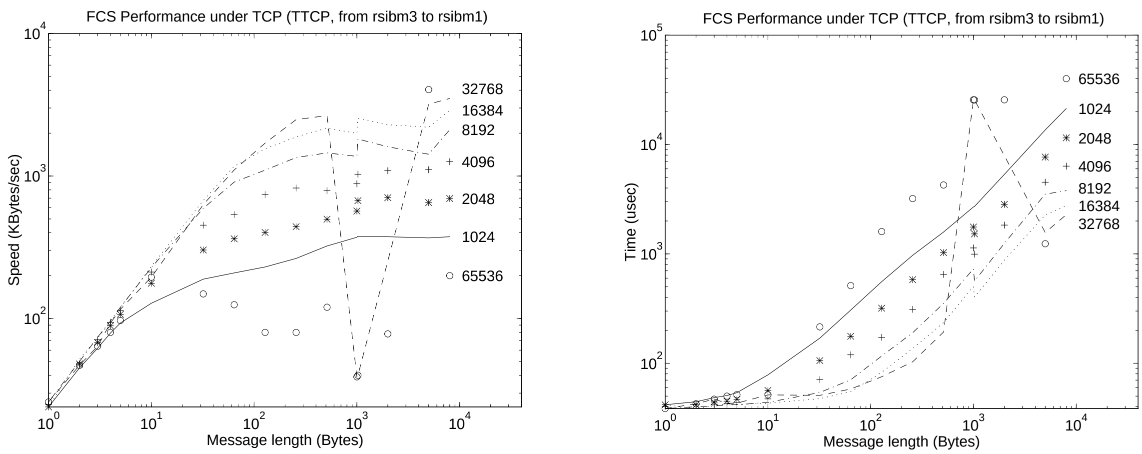


Figure 2a: Results obtained for transmission using *TCP* (Speed, Time).

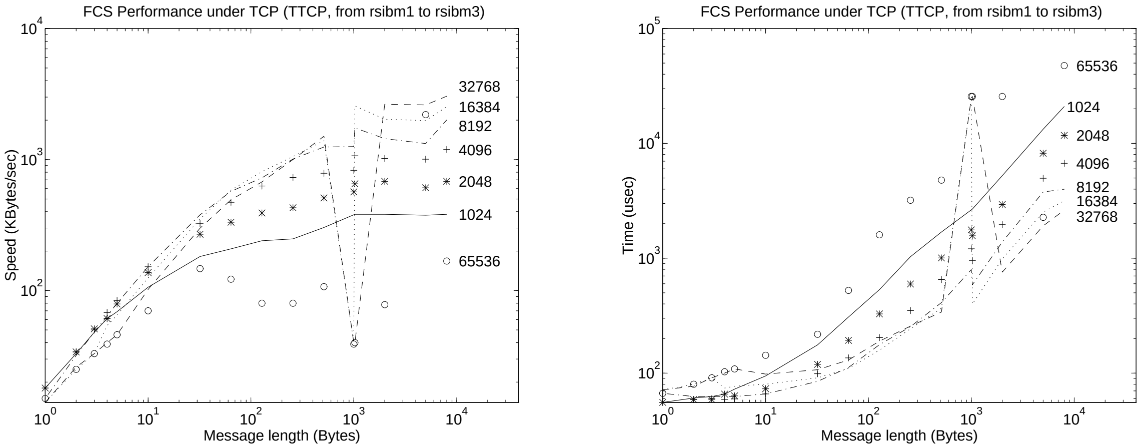


Figure 2b: Results obtained for transmission using *TCP* (Speed, Time).

From these graphs, we can conclude:

- the best performances have been obtained when the *RS/6000 590* is the sender and the *RS/6000 250* is the receiver (the *590* workstation is more powerful than the *250*).

- for both directions of communications, the shape of the measured performance chart (either speed or time) is alarming: it is not continuous. For instance, a drop of several hundreds of percent in the performance has been reported between sending a message of 900 bytes and a message of 1000 bytes (for socket buffer sizes greater than 8192 bytes). Moreover this result is completely deterministic: it can be reproduced at any time. We note that similar behaviour has been previously reported in a similar experiment using the same TCP implementation, with another type of medium (see appendix). In that case, for a given parameterization of the socket buffer size and the message size, a deadlock can occur between the two machines, and it is then deblocked by time expiry on the receiver side. We believe this is a likely explanation, which unfortunately can not be verified here, as no network analyser for Fibre Channel is yet available.

- the overhead of the system (interface plus fabric plus all the software), under *TCP/IP* is around 70 microseconds. The overhead is defined as the limit for the communication time when the message size to be transferred tends towards zero. For our application, this definition gives an over optimistic expectation of overhead for protocols which wait to merge messages before sending (see appendix). In that case, for instance instead of sending 10 times a 100 byte message, the system will send 5 times a 200 byte message.

III. Results using Direct Channel

We evolved and ran a couple of programs using Direct Channel interface. The principle of these programs is as follows: one writes some data in an infinite loop and the other one reads them in the same way. Time is reported on both sides as soon as the amount of data written (or read according to the side) exceeds a predefined value. Now, the maximum size of the message to be transferred is factory-limited to 65500 bytes. If the user desires to send a larger packet, it is the user's responsibility to manage the packet splitting on the sender side and the packet merging on the receiver side.

For these programs, there are two input parameters, which can be chosen independently:

- the size of the message to be sent,
- the number of times the message is sent (to perform an average measurement as previously).

The size of the message to be transferred ranges from 1 byte up to 65500 bytes.
The number of times the message is sent is equal to 10^4.

The read function of the driver can be used either in a blocking mode or in a non blocking mode. In a non blocking mode the function will return immediately with data if there is any. In a blocking mode, it will wait for up to 10 seconds for incoming data before returning.

It is important to note that the direct channel is of a streaming type. For example, if a read is issued for 1000 bytes, and there are only 10 bytes that have come in, the read function will return the 10 bytes only and set the number of bytes read to 10. If there is more data than is requested, only the amount requested will be returned and rest will be returned on subsequent reads. In other words, no data is ever thrown out. It is handled in a first in first out manner.

The write function is quite straightforward: a non blocking write will send the data without waiting for an acknowledgment from the adapter. A blocking write waits for and returns the success or failure of the write to the calling process.

A. Communication between two machines

The preliminary experiment we performed was to measure the communication time between two machines without the Fibre Channel switch (point to point connection) and to compare it to the one obtained with the switch. In both cases the performance is identical. The insertion of the switch does not interfere.

In all the charts displayed in this paragraph, the solid line represents the performance curve of the sender, whilst the dashed line represents the performance curve of the receiver. In many cases, these two curves are identical and therefore impossible to be distinguished.

Figures *a* display the speed performance, figures *b* the communication time, and figures *c* the percentage of *CPU* time consumed in the measured aggregate communication time.

1. Communications between *RS/6000 250* and *RS/6000 590* machines
    a) *Read*/*Write* in a non blocking mode

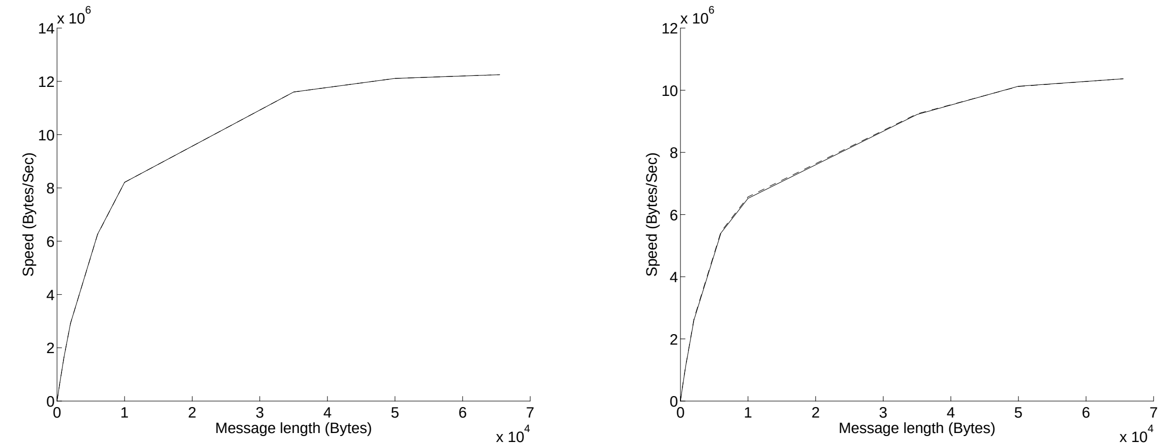


Figure 3a: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

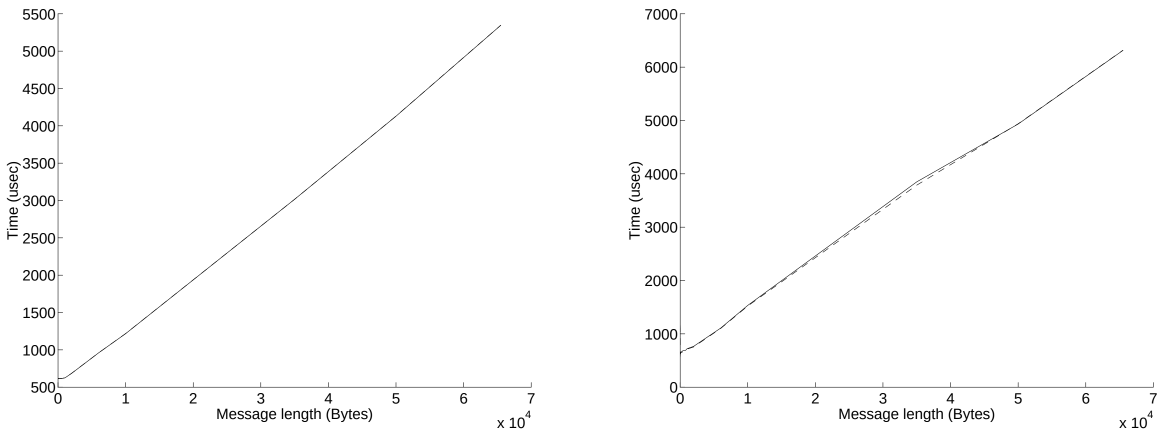


Figure 3b: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

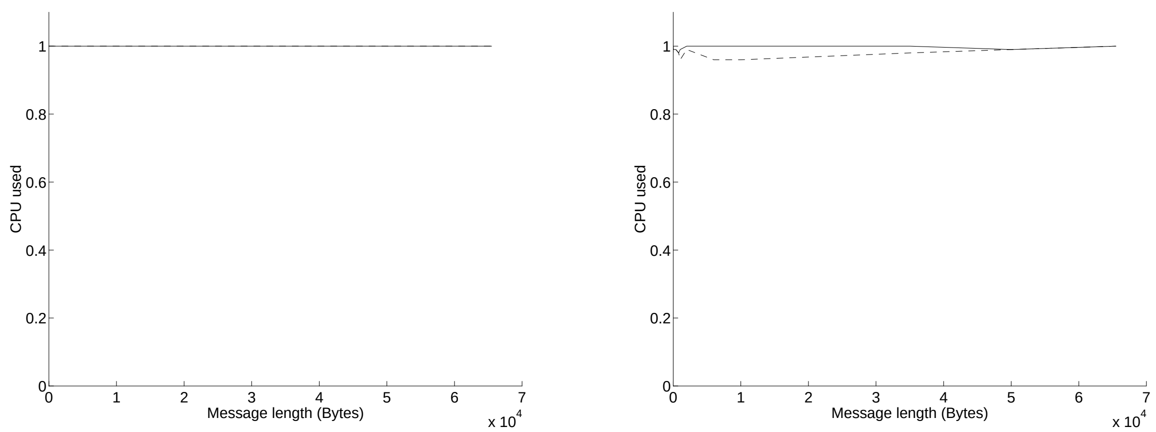


Figure 3c: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

b) *Write* in a non blocking mode, *Read* in a blocking mode

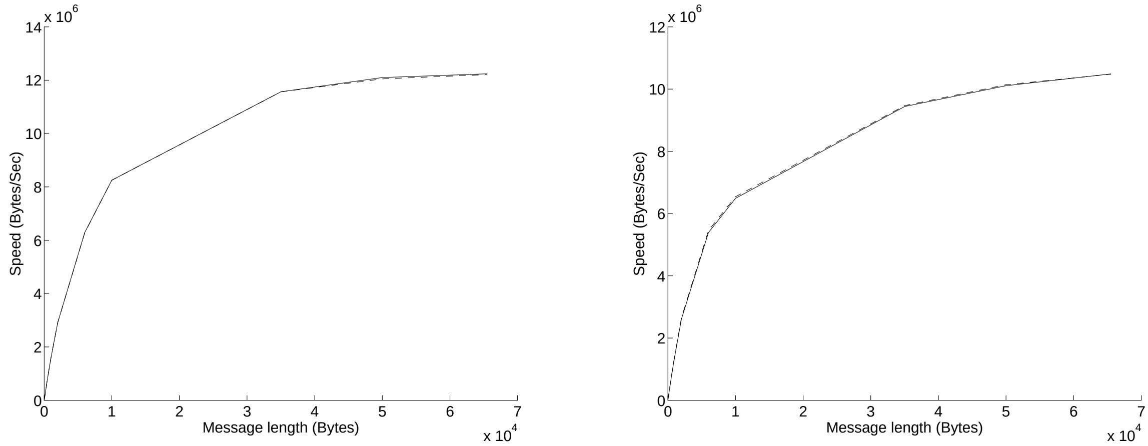


Figure 4a: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

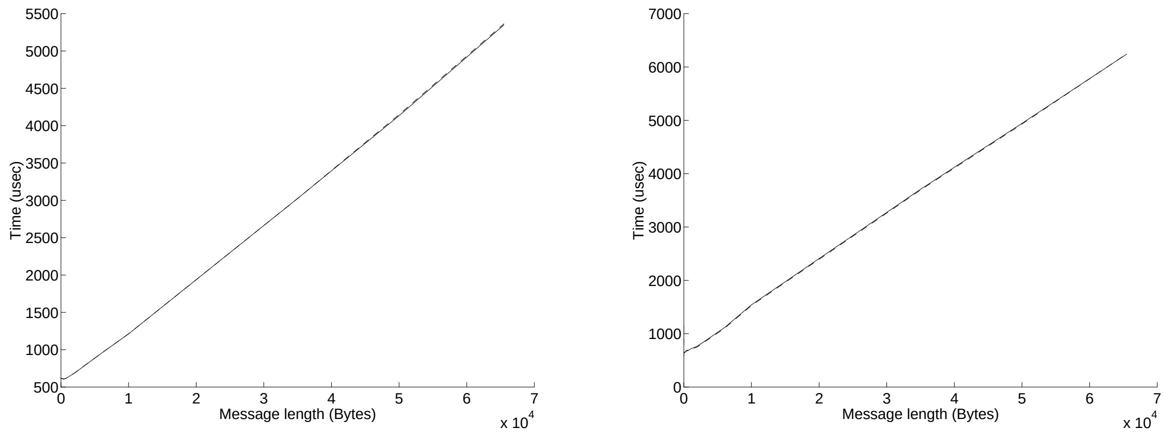


Figure 4b: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

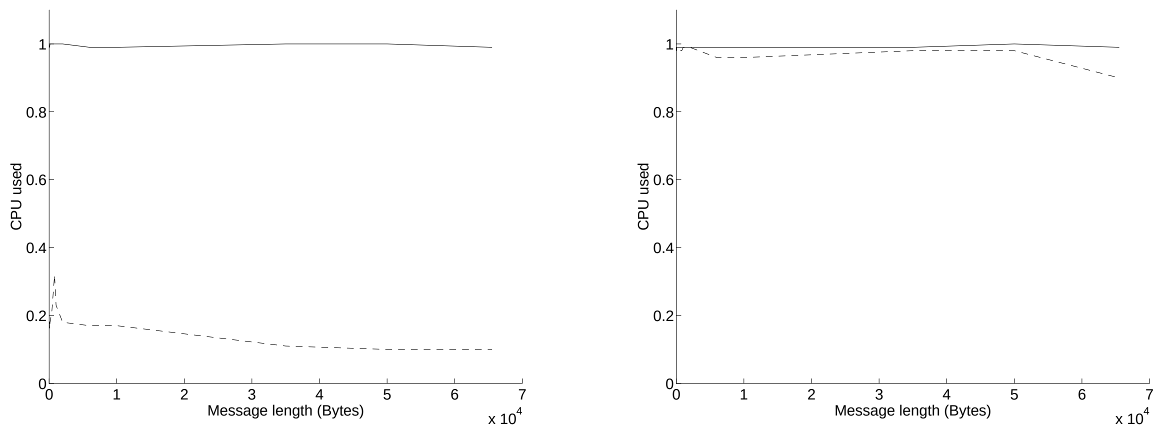


Figure 4c: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

c) *Write* in a blocking mode, *Read* in a non blocking mode

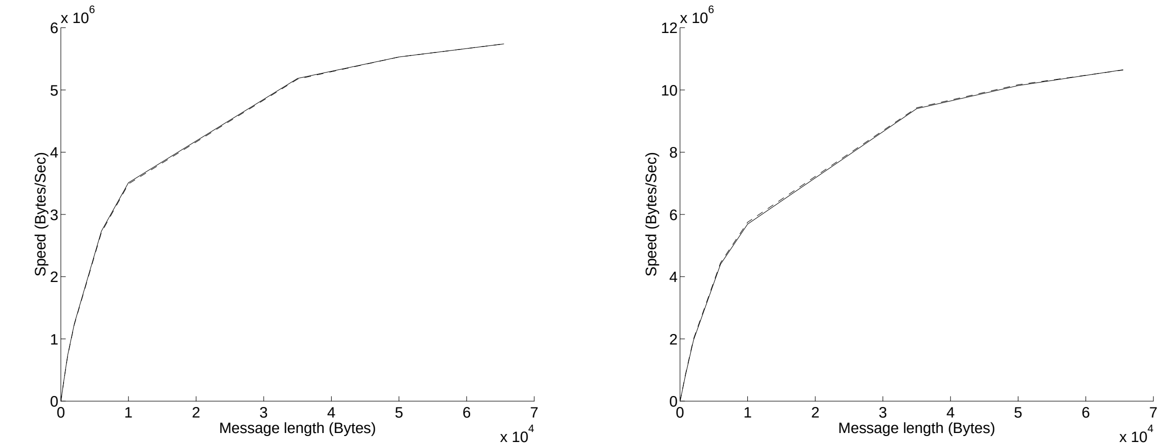


Figure 5a: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

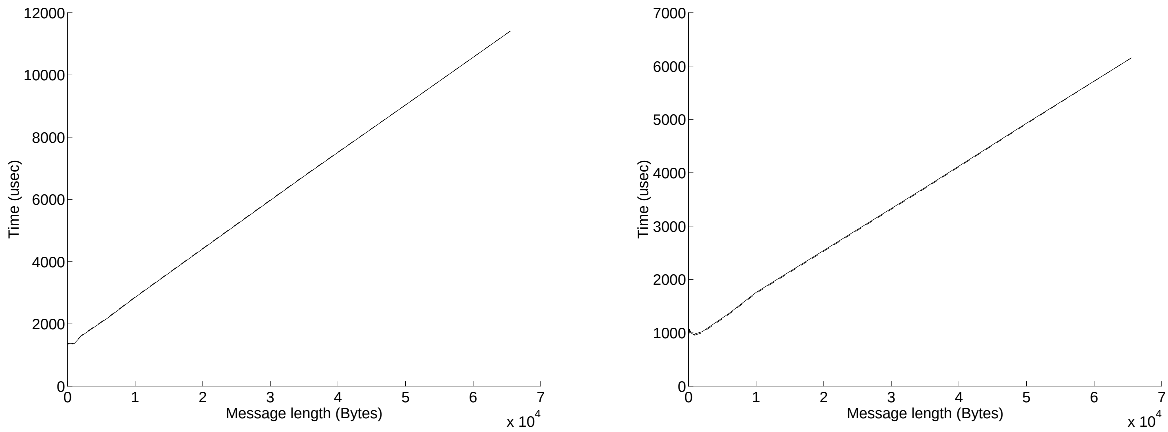


Figure 5b: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

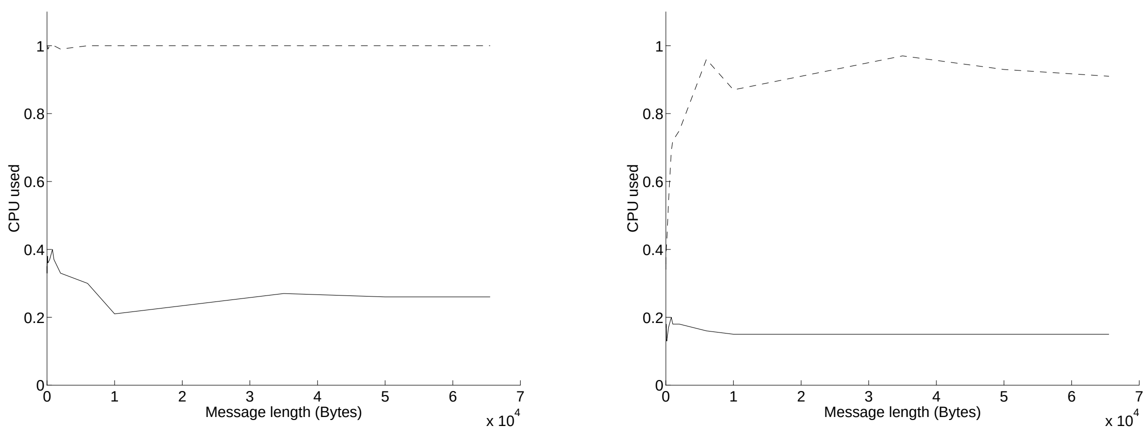


Figure 5c: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

d) *Read/Write* in a blocking mode

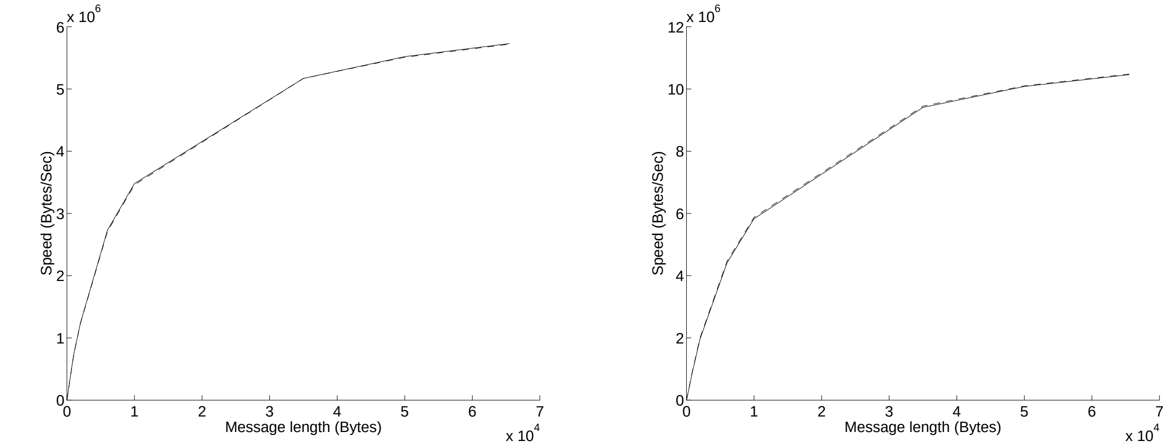


Figure 6a: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

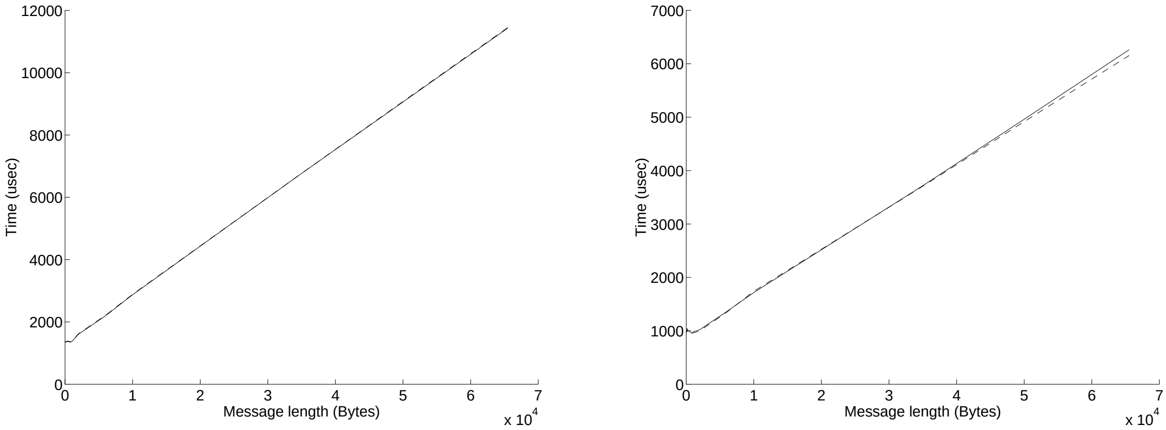


Figure 6b: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

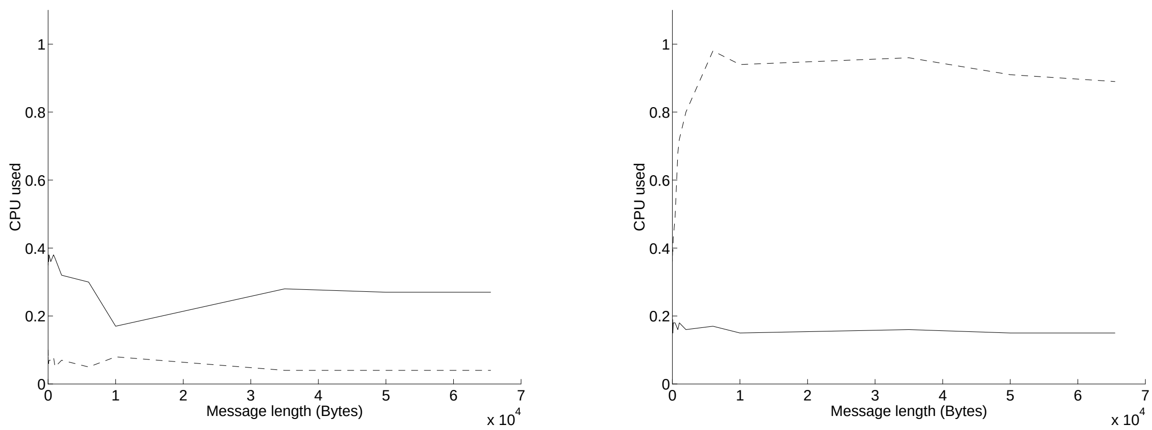


Figure 6c: (left) sender is *250*, receiver is *590*. (right) sender is *590*, receiver is *250*.

e) Conclusion

Let us just recall that our application is aimed at implementing the second level trigger according to a processor farm-based approach, each processor being connected to one or several Fibre Channel Fabrics, according to our requirements. In this case, each processor is really expected to do some processing task interleaving with communications. The results we present in this note are only related to communication performances, and therefore represent maximum values which never will be reached when implementing a parallel application in which some processing tasks will interleave the communication processes.

On figures 3, results have been achieved when using the *Read* and *Write* functions in a non blocking mode. The best performances are obtained when the most powerful workstation is receiving data (left graphs on figures *3*). We can infer that the execution of the *Read* function is slower than the one of the *Write* function when using non blocking mode. Under these conditions, the overhead of the system is around 615 microseconds, and the speed can reach 12.2 MBytes/s for transmitting 65500 byte messages.
The ratio between the *CPU* time and the communication time is around 100 percent. This is due to the nature of the algorithm: the *Write* and *Read* functions are executed in an endless loop. On the sender side, even if the buffer is full, the *Write* function will continue on attempting to send some data and will return an error message (overflow), until some free space will be available in the buffer. That explains why the processor is always busy.The same problem occurs on the receiver side.

On figures *4*, results have been achieved when using the *Read* function in a blocking mode and the *Write* function in a non blocking mode. As previously, the best performances are obtained when the most powerful workstation is receiving data (left graphs on figures *4*). The results are similar to those obtained on figures *3*: the execution of the *Read* function in a blocking mode takes indeed the same time as if it was achieved in a non blocking mode. Under these conditions, the overhead of the system is around 615 microseconds, and the speed can reach 12.2 MBytes/s for transmitting 65500 byte messages. On the left graph of figure *4c*, we can see that the percentage of CPU time used at the receiver side is below 20 percent: that means that the receiver is sustaining the data flow without any problem and must often wait for data to come in.

On figures *5*, results have been achieved when using the *Read* function in a non blocking mode and the *Write* function in a blocking mode. Now, the execution of the *Write* function takes a longer time than the one of the *Read* function, because it has to wait for an acknowledgment before returning. This explains why the best performances are obtained when the most powerful workstation is sending data (right graphs on figures *5*). Under these conditions, the overhead of the system is around 1 millisecond, and the speed can reach 10.6 MBytes/s for transmitting 65500 byte messages.

On figures *6*, results have been achieved when using the *Read* and *Write* functions in a blocking mode. As previously, the best performances are obtained when the most powerful workstation is sending data (right graphs on figures *6*). The results are similar to those obtained on figures *5*: the execution of the *Read* function in a blocking mode takes indeed less time than the execution of the *Write* function in a blocking mode. Under these conditions, the overhead of the system is around 1 millisecond, and the speed can reach 10.5 MBytes/s for transmitting 65500 byte messages.

2. Communications between *RS/6000 590* and *RS/6000 C10* machines

   a) *Read/Write* in a non blocking mode

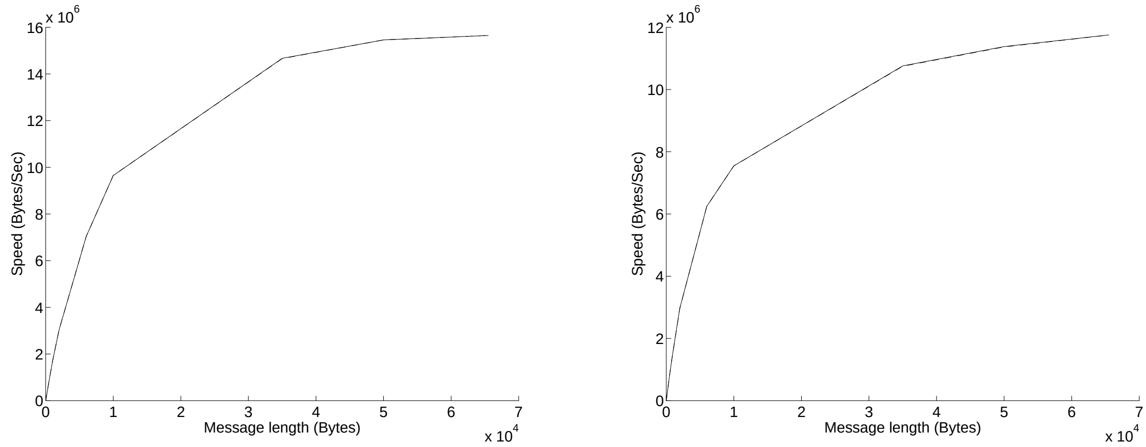


Figure 7a: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

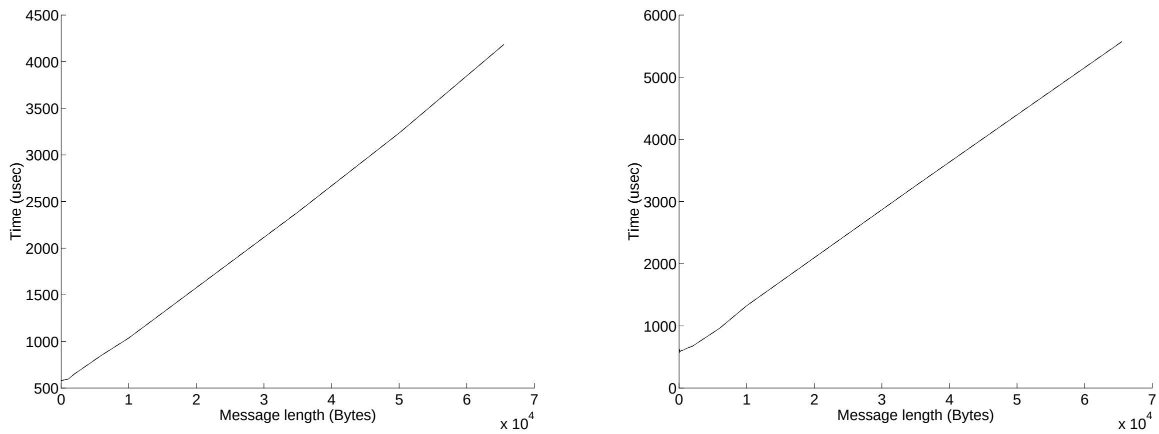


Figure 7b: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

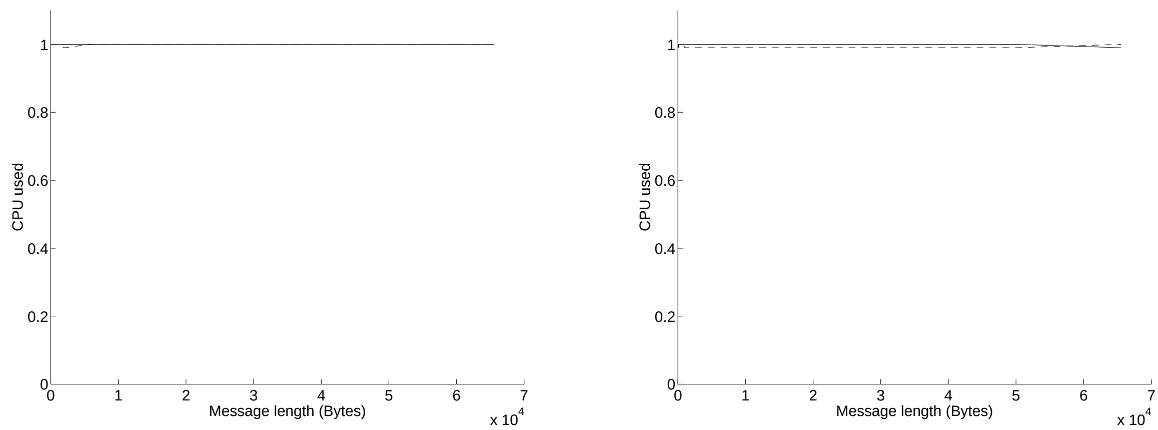


Figure 7c: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

b) Write in a non blocking mode, Read in a blocking mode

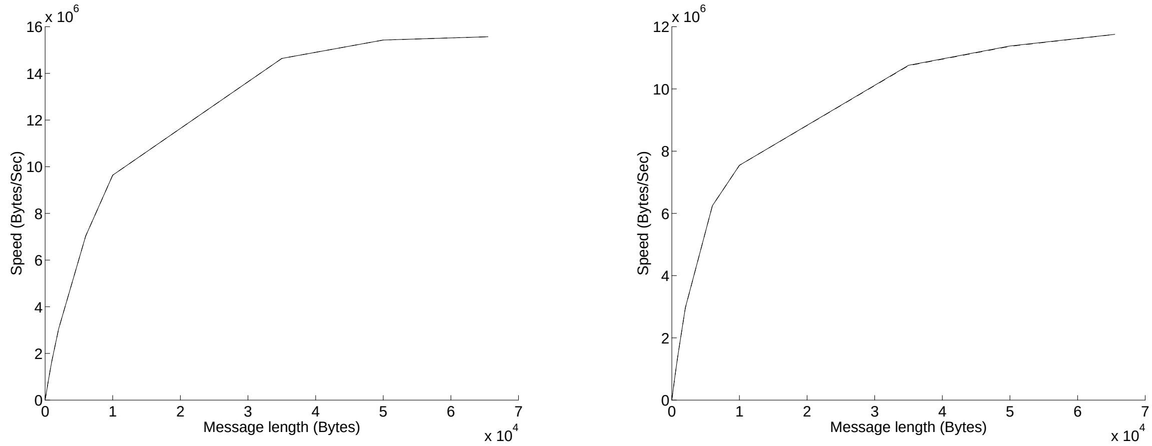


Figure 8a: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

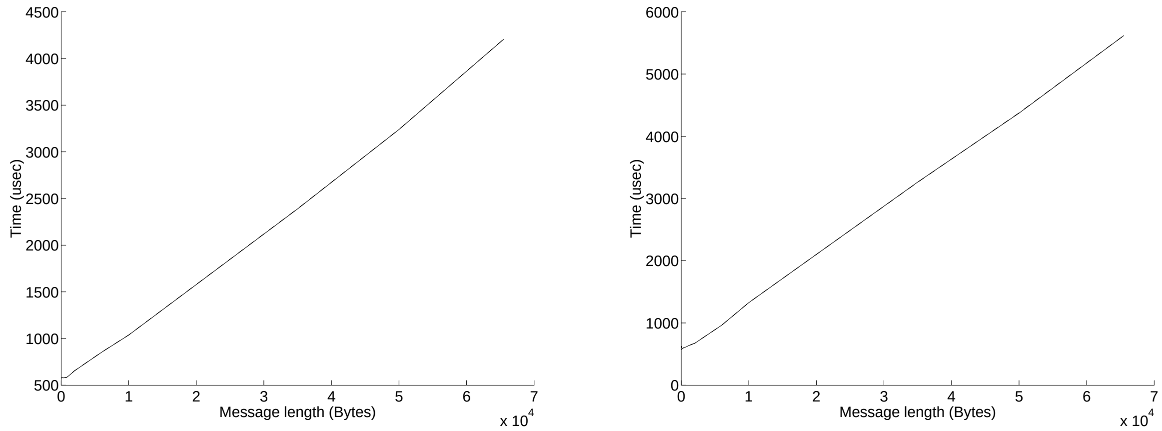


Figure 8b: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

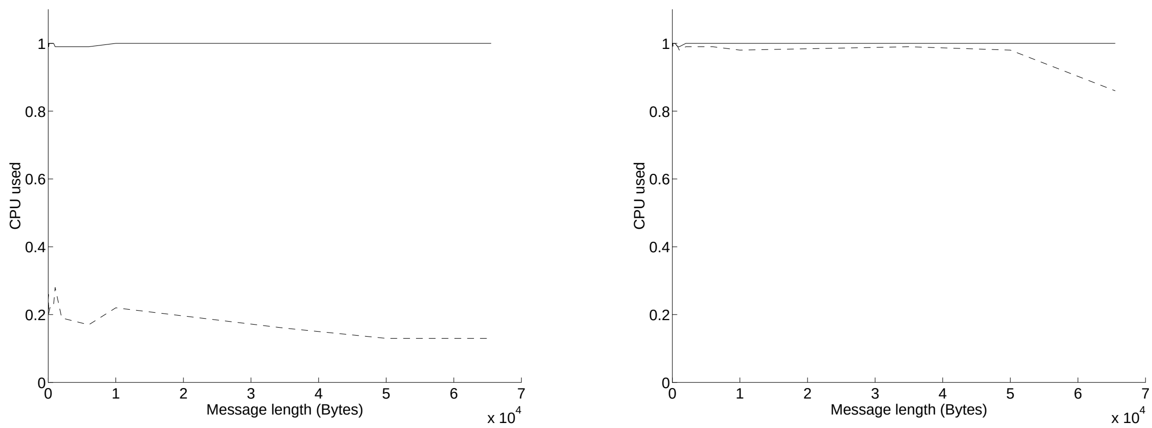
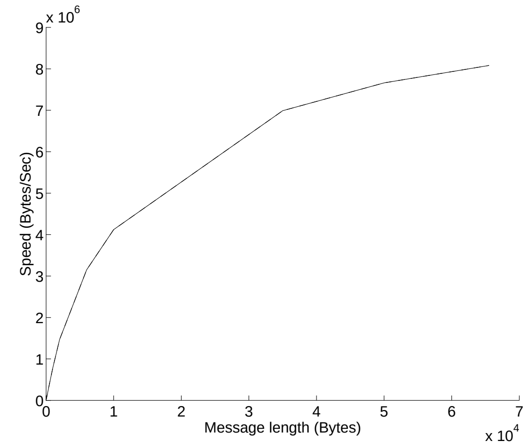


Figure 8c: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

c) *Write* in a blocking mode, *Read* in a non blocking mode



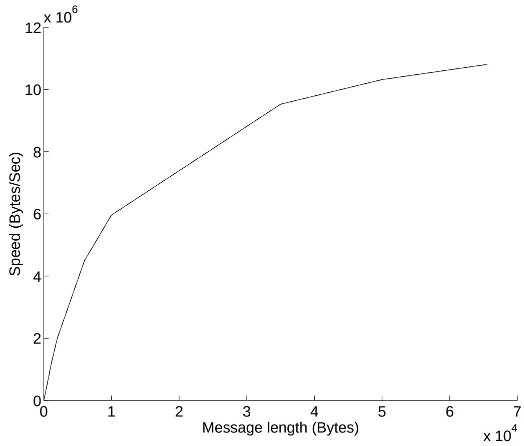
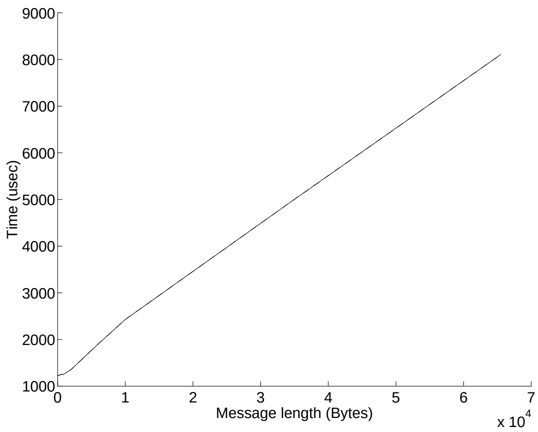


Figure 9a: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.



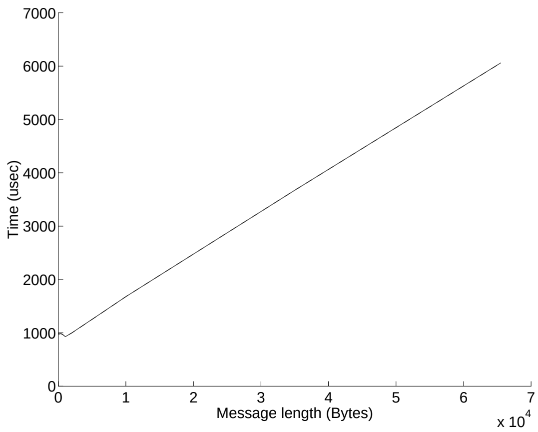
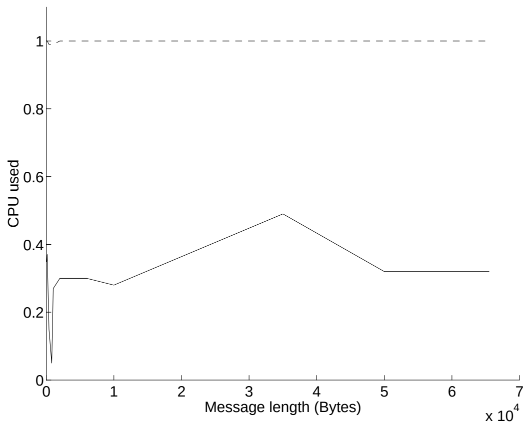


Figure 9b: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.



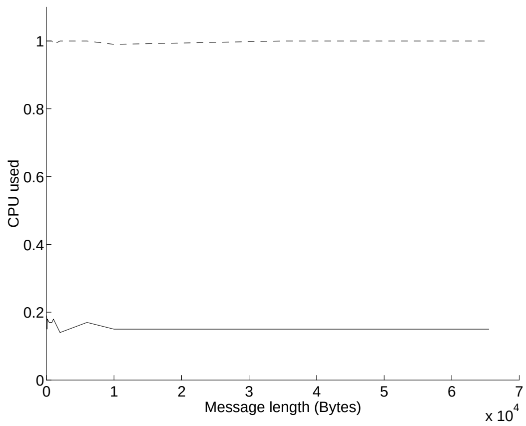


Figure 9c: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

d) *Read/Write* in a blocking mode

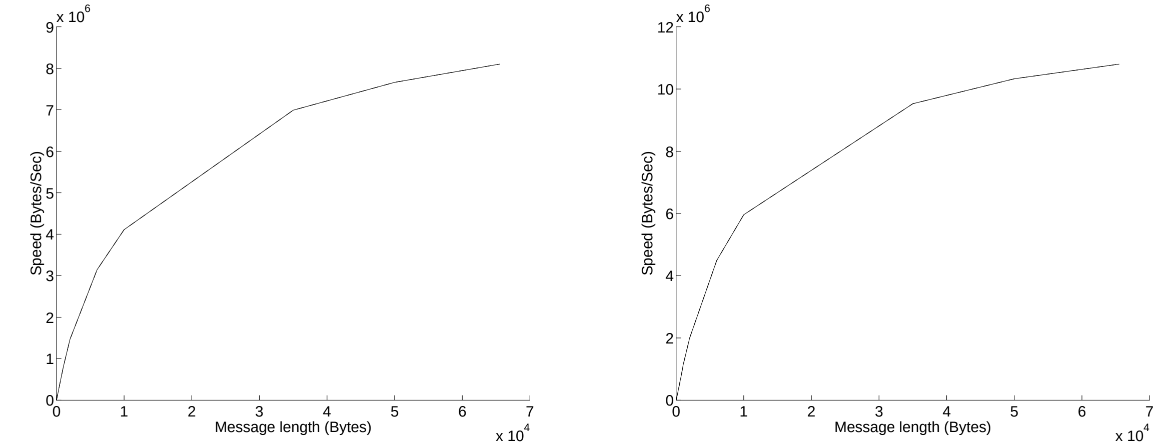


Figure 10a: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

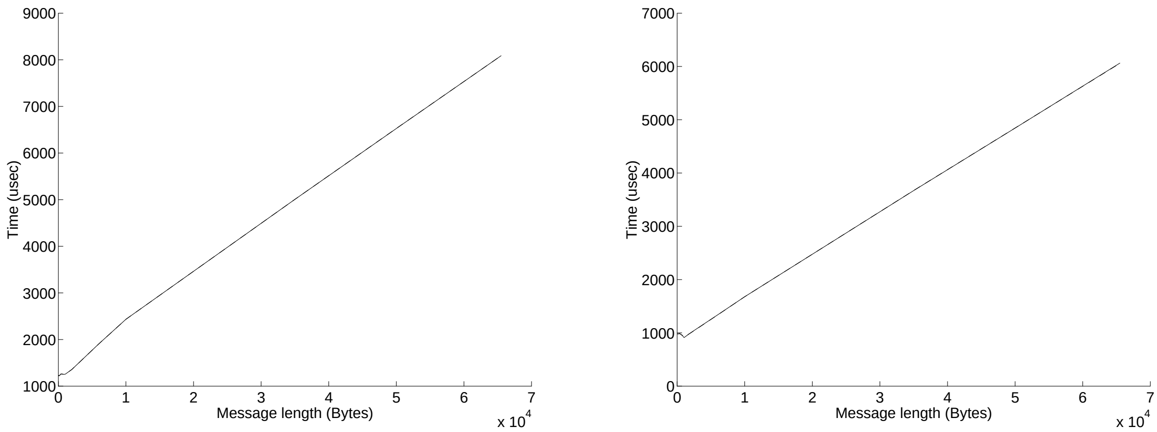


Figure 10b: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

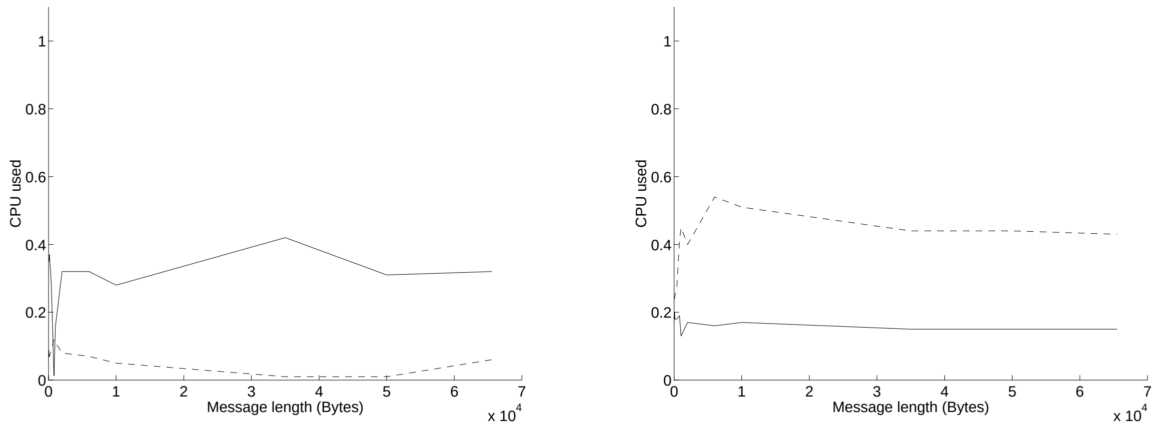


Figure 10c: (left) sender is *C10*, receiver is *590*. (right) sender is *590*, receiver is *C10*.

e) Conclusion

The concluding remarks made in paragraph *III.A.1.e.* are still valid here.

On figures *7*, results have been achieved when using the *Read* and *Write* functions in a non blocking mode. The best performances are obtained when the most powerful workstation is receiving data (left graphs on figures *7*), because the execution of the *Read* function is slower than the one of the *Write* function when using non blocking mode. Under these conditions, the overhead of the system is around 575 microseconds, and the speed can reach 15.6 MBytes/s for transmitting 65500 byte messages. Comparing these results to those presented on figures *3* points out the performance improvement induced by the replacement of the *250* workstation by the *C10*.

On figures *8*, results have been achieved when using the *Read* function in a blocking mode and the *Write* function in a non blocking mode. As previously, the best performances are obtained when the most powerful workstation is receiving data (left graphs on figures *8*). The results are similar to those obtained on figures *7*: the execution of the *Read* function in a blocking mode takes indeed the same time as if it was achieved in a non blocking mode. On the left graph of figure *8c*, we can see that the percentage of CPU time used at the receiver side is below 20 percent: that means that the receiver is sustaining the data flow without any problem and must often wait for data to come in.

On figures *9*, results have been achieved when using the *Read* function in a non blocking mode and the *Write* function in a blocking mode. Now, the execution of the *Write* function takes a longer time than the one of the *Read* function. This explains why the best performances are obtained when the most powerful workstation is sending data (right graphs on figures *9*). Under these conditions, the overhead of the system is around 980 microseconds, and the speed can reach 10.8 MBytes/s for transmitting 65500 byte messages.

On figures *10*, results have been achieved when using the *Read* and *Write* functions in a blocking mode. As previously, the best performances are obtained when the most powerful workstation is sending data (right graphs on figures *10*). The results are similar to those obtained on figures *9*: the execution of the *Read* function in a blocking mode takes indeed less time than the execution of the *Write* function in a blocking mode. Under these conditions, the overhead of the system is around 980 microseconds, and the speed can reach 10.8 MBytes/s for transmitting 65500 byte messages.

B. Communications between three machines

We performed two kind of experiments based on three active machines connected around the switch. The first one consists of one machine (the most powerful) acting as a sender and the two other machines acting as receivers alternatively. The second one consists of one machine (always the most powerful) acting as a receiver and two machines acting as senders alternatively. For both experiments, *Read*/*Write* in a non blocking mode and in a blocking mode have been tested and results are reported hereinafter.

For all the charts displayed in this paragraph, the legend is as follows:

- case of two senders and a receiver:

- the dashdot line represents the performance curve of the *sender1*
- the star line represents the theoretical performance curve of the *sender1*
- the solid line represents the performance curve of the *sender2*
- the circle line represents the theoretical performance line of the *sender2*
- the dashed line represents the performance curve of the *receiver*
- the plus line represents the theoretical performance curve of the *receiver*
- the dotted line represents the maximum theoretical speed of an optic fibre

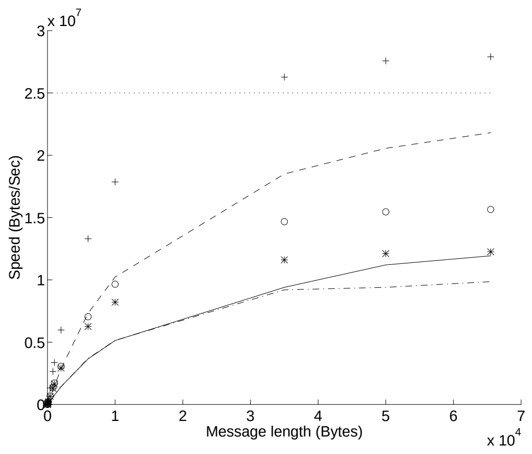
- case of two receivers and a sender:

- the dashdot line represents the performance curve of the *receiver1*
- the star line represents the theoretical performance curve of the *receiver1*
- the solid line represents the performance curve of the *receiver2*
- the circle line represents the theoretical performance line of the *receiver2*
- the dashed line represents the performance curve of the *sender*
- the plus line represents the theoretical performance curve of the *sender*

The theoretical performance curve of the *sender1* (respectively *sender2*) was obtained by measuring the performance of the *sender1* (respectively *sender2*) when transferring some data to the *receiver*, without any other machine interfering on the network (results of paragraph *III.A.*). The theoretical performance curve of the *receiver* has been inferred from the theoretical performance curves of the *sender1* and the *sender2*. The maximum theoretical speed of an optic fibre was indicated only on one graph to recall the uppermost limit towards which the experimental results could tend.
Similarly, the theoretical performance curve of the *receiver1* (respectively *receiver2*) was obtained by measuring the performance of the *receiver1* (respectively *receiver2*) when receiving some data from the *sender*, without any other machine interfering on the network.

As previously for communications between two machines, figures *a* display the speed performance, figures *b* the communication time, and figures *c* the percentage of *CPU* time consumed in the measured aggregate communication time.

1. *Read/Write* in a non blocking mode



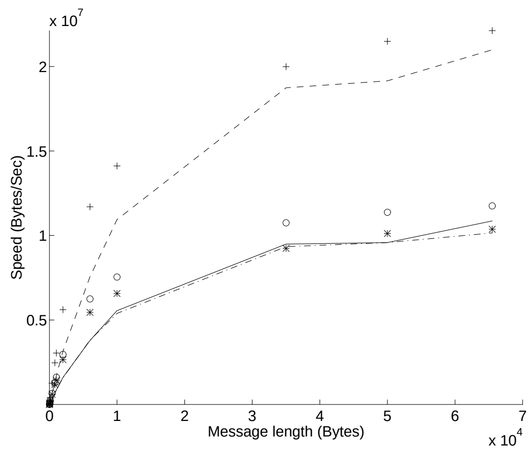


Figure 11a: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

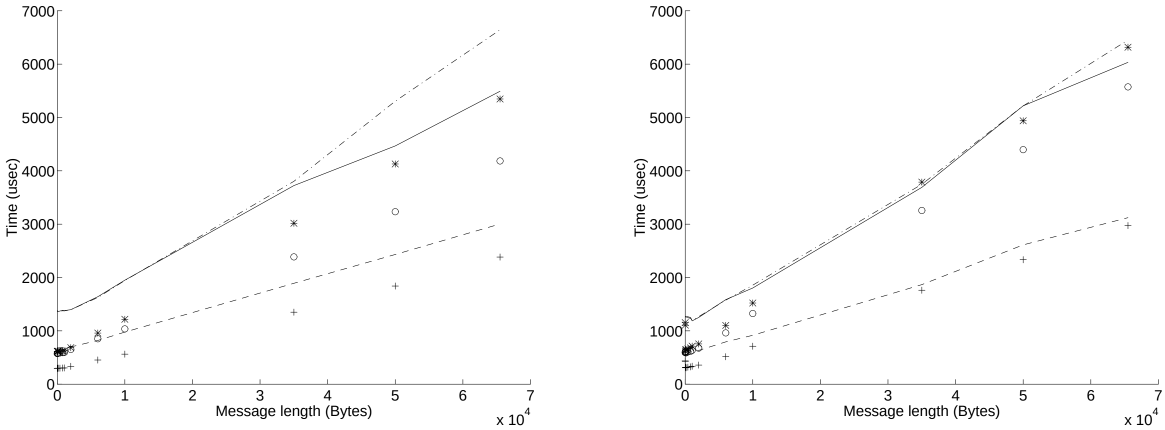


Figure 11b: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

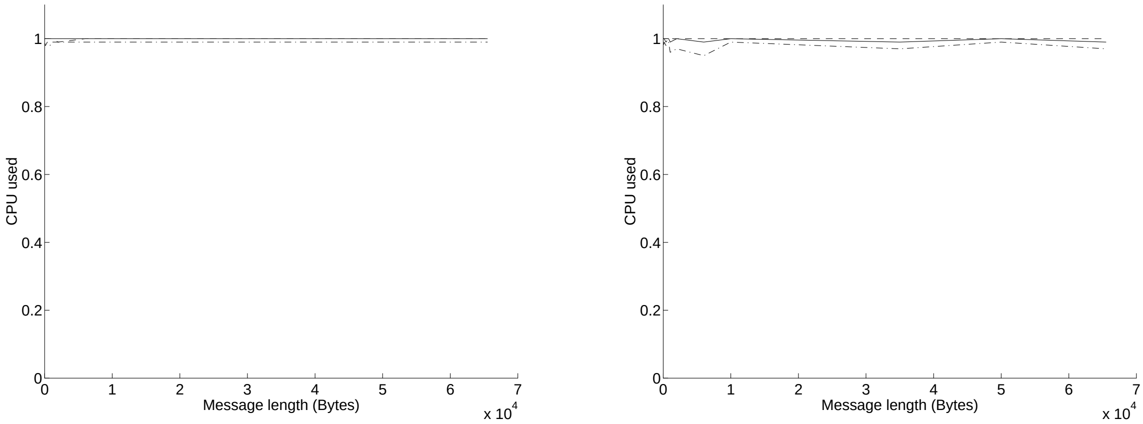


Figure 11c: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

2. *Read*/*Write* in a blocking mode

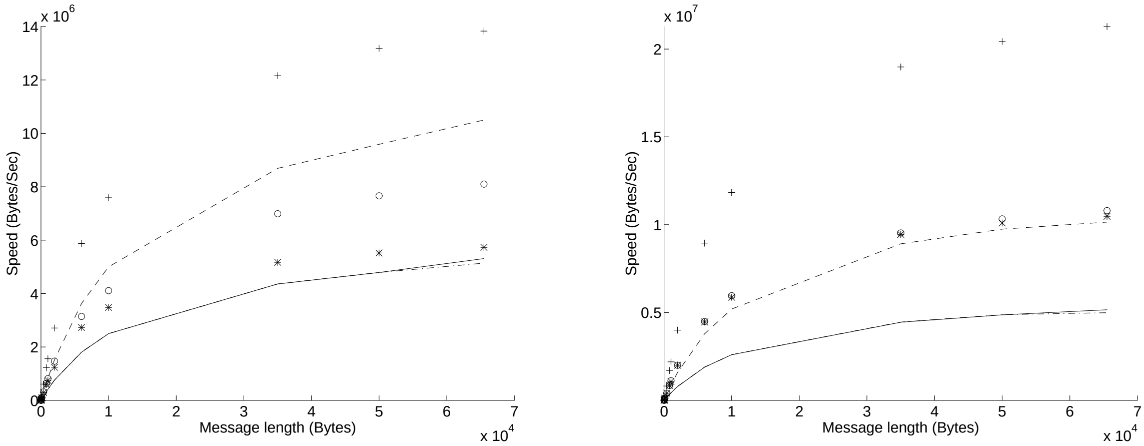


Figure 12a: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

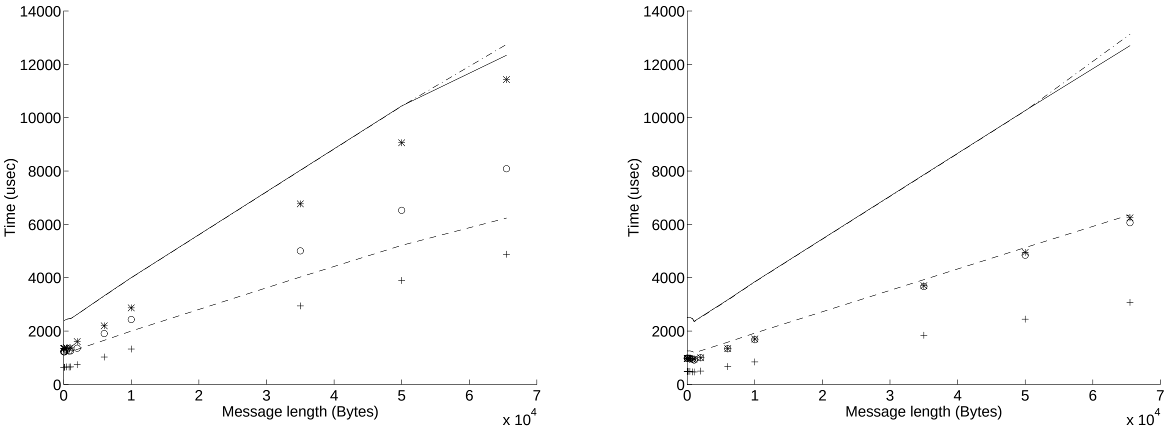


Figure 12b: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

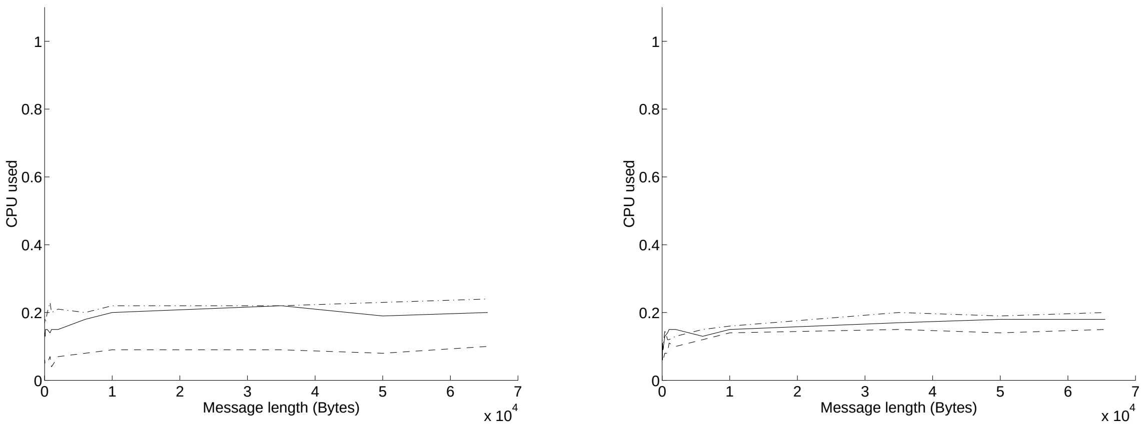


Figure 12c: (left) *sender1* is *250*, *sender2* is *C10*, *receiver* is *590*. (right) *sender* is *590*, *receiver1* is *250*, *receiver2* is *C10*.

3. Conclusion

On figures *11*, results have been achieved when using the *Read* and *Write* functions in a non blocking mode. When the most powerful workstation acts as a receiver (respectively sender), the measured performances on the receiver side (respectively sender side) are as follows:

- overhead of 680 microseconds (respectively 630 microseconds),
- speed of 21.8 MBytes/s (respectively 21 MBytes/s) for receiving 65500 byte messages.

On figures *12*, results have been achieved when using the *Read* and *Write* functions in a blocking mode. When the most powerful workstation acts as a receiver (respectively sender), the measured performances on the receiver side (respectively sender side) are as follows:

- overhead of 1200 microseconds (respectively 1250 microseconds),
- speed of 10.5 MBytes/s (respectively 10.1 MBytes/s) for receiving 65500 byte messages.

Now, for the two experiments, one can remark that the aggregate performance is improved, but only for messages with a size greater than 2000 bytes.
It looks like if the most powerful workstation is not enough powerful for maintaining the data flow enforced by the two other machines. Otherwise, if it would be capable of sustaining the sum of each machine's data flow, the practical performances would fit the theoretical ones (see figures *11* and *12*).

IV. Conclusion

The best performances, as expected, are obtained when using the Direct Channel interface of the driver. Hereafter are summarized the best results measured, according to the hardware configuration used:

- communications between RS/6000 250 and RS/6000 590 workstations: the overhead of the system is around 615 microseconds, and the speed can reach 12.2 MBytes/s for transmitting 65500 byte messages (*Read* and *Write* functions are implemented in a non blocking mode).

- communications between RS/6000 C10 and RS/6000 590: the overhead of the system is around 575 microseconds, and the speed can reach 15.6 MBytes/s for transmitting 65500 byte messages (*Read* and *Write* functions are implemented in a non blocking mode).

- communications between RS/6000 250, RS/6000 C10 and RS/6000 590 workstations: when the most powerful workstation acts as a receiver (respectively sender), on the receiver side (respectively sender side), the overhead is around 680 microseconds (respectively 630 microseconds) and the speed can reach 21.8 MBytes/s (respectively 21 MBytes/s) for 65500 byte messages (*Read* and *Write* functions are implemented in a non blocking mode).

All the results presented in this note represent maximum reachable values when implementing a parallel application in which some processing tasks will interleave the communication processes.
For the project in which we are involved (implementation of second level trigger), the size of messages to be transferred has been defined to be either 64 bytes or 1024 bytes. In this case, the corresponding performance of the system is located in left parts of graphs, where the lowest results are reported. Assuming the workstations could be *RS/6000 C10* and *590* models for instance, the best communication time between two of these machines, for a 64 byte message will be around 580 microseconds and 600 microseconds for a 1024 byte message.
The next step in studying the behaviour of the Fiber Channel Fabric will be to connect several other workstations around it, and to estimate several important parameters like for instance the interference induced by the switch when several independent communications are performed in parallel, the way contention problems are managed, ...

V. References

[1] Fibre Channel Association: “Fibre Channel: Connection to the Future”, ISBN: 1-878707-19-1, 1994
[2] “CXT 250 16-port Switch”, Installer’s/User’s manual, Publication N°. PUB 007 A Rev. D.
[3] “MCA CIM 250”, Installer’s/User’s Manual, Publication N° PUB 009 A Rev. Draft.
[4] “Direct Channel Interface”, User’s Specification, Part N° DOC 013 Rev. A.
[5] “Description of a problem which occurs when using TCP/IP to transfer bulk data between two machines“, Danny Davids, CERN note in progress.

VI. Appendix: Problem reported when using *TCP/IP* to transfer data

D. Davids reported a problem of deadlock which can be reproduced with many types of machines and is independent of the physical medium on which the *TCP* transfer is performed.
The program used to test out the performance of data transmission was ‘ttcp’ (the same as we used to obtain results in paragraph *II*). The experiment was achieved between heterogeneous machines (a *RS/6000* IBM workstation and a non IBM workstation). Owing to a network analyser, one was able to see what was really happening during the communication between the two machines.
Under the same conditions of use (message size of 2048 bytes, socket buffer size of 8192 bytes, message sent 1000 times), tests have been performed between the following platforms, and a surprisingly low transmission speed has been measured:

Sender: HP-725 ATM (MMF) ---> Receiver: RS/6000-530 ATM (MMF) 20.5 Kbps
Sender: HP-725 CDDI (UTP5) ---> Receiver: RS/6000-530 FDDI (MMF) 30.5 Kbps
Sender: ULTRIX FDDI (MMF) ---> Receiver: RS/6000-530 FDDI (MMF) 30.5 Kbps

The trace achieved by the network analyser is given below:

Delta Time DATA---> Contents of the DATA packet --->
in Seconds <----ACK <---- Contents of the ACK packet

DEC,HP ---> IBM ACK=179200002 SEQ=6841346 LEN=2048 WIN=8192
0.0022 DEC,HP ---> IBM ACK=179200002 SEQ=6843394 LEN=4096 WIN=8192
0.1952 DEC,HP <--- IBM ACK=6847490 WIN=16384
0.0017 DEC,HP ---> IBM ACK=179200002 SEQ=6847490 LEN=2048 WIN=8192
0.0019 DEC,HP ---> IBM ACK=179200002 SEQ=6849538 LEN=4096 WIN=8192
0.1969 DEC,HP <--- IBM ACK=6853634 WIN=16384
0.0019 DEC,HP ---> IBM ACK=179200002 SEQ=6853634 LEN=2048 WIN=8192
0.0014 DEC,HP ---> IBM ACK=179200002 SEQ=6855682 LEN=4096 WIN=8192
0.1966 DEC,HP <--- IBM ACK=6859778 WIN=16384
0.0019 DEC,HP ---> IBM ACK=179200002 SEQ=6859778 LEN=2048 WIN=8192
0.0015 DEC,HP ---> IBM ACK=179200002 SEQ=6861826 LEN=4096 WIN=8192
0.1965 DEC,HP <--- IBM ACK=6865922 WIN=16384
0.0014 DEC,HP ---> IBM ACK=179200002 SEQ=6865922 LEN=2048 WIN=8192
0.0018 DEC,HP ---> IBM ACK=179200002 SEQ=6867970 LEN=4096 WIN=8192
0.1975 DEC,HP <--- IBM ACK=6872066 WIN=16384
0.0013 DEC,HP ---> IBM ACK=179200002 SEQ=6872066 LEN=2048 WIN=8192

0.0016 DEC,HP ---> IBM ACK=179200002 SEQ=6874114 LEN=4096 WIN=8192
0.1964 DEC,HP <--- IBM ACK=6878210 WIN=16384

In this trace, one can easily see the result of the *TCP* implementation: the sender first sends 2 Kbytes of data, then it waits until it is able to send 4 KBytes, then it waits for an acknowledgment as it wants to send again 4 KBytes instead of 2 KBytes. One can also see that the receiver decides to wait for other data packets before sending its acknowledgment packet. So a deadlock occurs and it is deblocked only after a time expiry of 200 milliseconds.
One can also remark that the size of the message to be transferred was set to 2048 bytes, but in many cases, two messages were grouped together to form a message of 4096 bytes to be sent.
For more details, see [5], which is a note currently being written by D. Davids.