LA-UR -85-3583

CONF-8510194--4

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--85-3583

DE86 002396

TITLE DATA ACQUISITION AND CONTROL USING ETHERNET

AUTHOR(S) E. Paul Elkins, AT-4

SUBMITTED TO 2nd International Workshop on Accelerator Control Systems
Los Alamos, New Mexico, October 7-10, 1985

DISCLAIMER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

DATA ACQUISITION AND CONTROL USING ETHERNET*

E. PAUL ELKINS, AT-4, MS H821
Los Alamos National Laboratory, Los Alamos, NM 87545

We have developed a distributed computer control system to monitor and control a linear accelerator. This system consists of two PDP-11s and eight LSI 11/23s linked together with ETHERNET. The higher level systems (control consoles, etc.) use the RSX11M operating system, whereas the data acquisition and control is performed using the RSX11S operating system downline loaded from a central host computer. Locally written ETHERNET drivers are used to reduce the CPU overhead and therefore improve system response. The ETHERNET system permits remote file access by means of operator or program interaction, as well as supporting downline system loading. Control-system functions supported are supervisory control, closed-loop control, data monitoring, and data recording.

1. Introduction

The original control system [1] for the Fusion Materials Irradiation Test Facility (FMIT) was a star-configured computing network consisting of two distinct levels of computing. The upper level consisted of control and number-crunching computers (CPU) running the RSX11M operating system, whereas the lower level was the data acquisition layer and ran the RSX11S operating system. In the original system there were three CPUs in the upper layer; one was a central control CPU (Primary), which was a PDP-11/60 located at the center of the star, whose function was to drive the two main control consoles. The second CPU, also a PDP-11/60, was used as the software development machine (Support) and the third CPU, an LSI-11/23, performed on-line beam reconstruction (RC) analysis.

All data acquisition CPUs (DAC) were LSI-11/23s and were housed in CAMAC [2], which was used as the primary data acquisition and control hardware. In addition, all of these CPUs were downline loaded from the Primary through DECNET over 56-kilobit lines. There were five DACs total, four for the different accelerator systems and a hardware/software test CPU.

*Work supported by the US Dept. of Energy.

Each main control console consisted of a color-graphics scope; a knob panel (with four control knobs) and its associated display (plasma panel); and a color alarm-message scope. The graphics scope and the plasma panel were both touch sensitive. All of the above devices were interfaced through an RS232 serial interface except for the graphics scope, which had a parallel-byte transfer bus using programmed I/O. There was also a bank of 16 fixed buttons and lights interfaced to the console CPU used for clearing display screens, selecting its Home display, etc.

The graphics scope was used to display menus, schematics, tabular data, and data plots, with all operator interaction through the touch panel. The knobs were software assignable to remote equipment and were used to adjust this equipment while its associated plasma panel displayed data. Touch commands on the plasma panel were used to turn remote equipment on or off; using all of the console features listed above, an operator can monitor and control the entire accelerator from the central control room.

All the DACs have a set of common programs plus some that were unique to each node. The common programs were a surveillance program (limit checker), a local console program, and a general-purpose equipment module for processing equipment or channel commands. In addition, each node had the required process modules to perform its assigned task; that is, the cooling node controlled the cooling for the accelerator components such as the radio-frequency quadrupole (RFQ), etc.

All communication between nodes was done using a DECNET interface to a message concentrator in each node. The application codes all communicated with the message concentrator through formatted packets; hence, the actual DECNET communication between nodes was done by the message concentrator.

2. System upgrade

This system became operational during 1982, and late in the year it became apparent that the system performance was not as good as expected. The system was sluggish, but usable, with one console running; however, the system was intolerably slow when running two consoles. Because accelerator operations required running two consoles simultaneously, something had to be done to improve the overall system response.

The first step was to analyze the system and try to isolate the problem areas. Four factors seemed to limit the system performance:

(1) The high DECNET CPU load

(2) Too much interrupt per character I/O

(3) Not enough memory

(4) Lack of system flexibility

We could have opted for a higher performance CPU in place of the Primary. However, we felt that this would not give the increased system performance required; also, such a configuration was very expensive. It seemed the best solution was to find a way to reduce the CPU overhead and to improve system flexibility. System flexibility could be improved by changing the network architecture from a star to a bus structure. A measure of system flexibility is the ease with which a new node can be added or removed from the network. This capability may be desirable if there is a change in requirements or if there is a computer-overload condition. At about this time, a local area network (LAN) called ETHERNET [3] was beginning to look like a viable option.

It appeared that by getting the right ETHERNET hardware and by writing an efficient software driver, we could satisfy Items (1) and (4) above. The obvious solution to Item (2) would be to use DMA RS232 hardware for all character output. A good solution to Item (3) would be to buy the newer, smaller, cheaper CPUs now available, which also have more memory, and use them as the console computer, that is, one per console position.

The best option was to buy two micro PDP-11/23s for the console CPUs and ETHERNET hardware for the other existing CPUs. This configuration would allow the continued use of all existing CPUs with only minor software changes. The highest priority was to develop a suitable driver for the ETHERNET controller and the necessary interface software to simplify the application code conversion. The proposed system [4], a reconfigured version of the original star network, is shown in fig. 1.

3. ETHERNET driver

Two drivers were written for the ETHERNET hardware, one for the RSX11M/RSX11S operating system and one for the stand-alone systems used on single-board computers (SBC) such as the FALCON (SBC-11/21). Both drivers used the same formatted packet.

The RSX driver responds to the standard QIO mechanism, and at present has four functions: send a packet, receive a packet, get link status, and get and clear link status. When a task sends a packet, it is transferred through DMA directly from its memory space. However, all received packets must be temporarily stored in the driver's buffer space until the real owner is determined, at which time the packet is transferred to the owner task's buffer space. The get-link status obtains for a task certain information about the controller such as node address and error counters. Most tasks only want their node address, which is available as a single word. The clear version of this request is used only by network management programs.

Any tradeoffs made in the driver design were made in the interest of speed and efficiency of data transfer. The stand-alone driver is of course, faster than the RSX driver because its operating system overhead is lower, but both drivers are functionally equivalent; therefore, either can communicate with the other. Both drivers keep a count of all errors, as well as all good packets sent and received.

4. System service

In a distributed control system, certain system services are required to make the network truly useful. The following system services are required, or at least are very desirable for most control systems.

Program-to-program communication

Downline system loading

Operator file transfer

Remote file access

Program-to-program communication (DATAGRAM) is of course required for any control system network, and the network should also be easy to use. The RSX DATAGRAM service uses the standard QIO system directive; however, application programs access this directive through a user-interface routine.

Downline system loading is required for most memory-resident systems, although some are EPROM based (FALCON system for example). However, even the FALCON systems downline loads its database, or (during development) the entire system is downline loaded. Downline loading requires the cooperation of at least one program in each CPU; this system uses two programs in the host CPU. The loading process is initiated at the remote node, either directly or

indirectly from an EPROM-based code called the primary loader. The primary loader sizes memory and requests the initial load from the host CPU. A fresh copy of the loader task is spawned so that multiple nodes can be loaded simultaneously. At the completion of the load process, the remote system is automatically executed.

Operator file transfer is provided, allowing an operator at one node to transfer files to or from a remote node. The SYNTAX is a subset of DECNET and an example of transferring a file to a local node follows:

```
NFC =NODE::[UIC]FILENAME.EXT
```

A file can be transferred to a remote node by entering the following:

```
NFC NODE::=[UIC]FILENAME.EXT
```

Remote file access is a by-product of the operator file-transfer system's service referred to earlier. It allows a program to read, write, or create files in a remote node. Remote access can be by sequential-, direct-, or block-mode method of access. Again, when a remote-file operation is requested, a fresh copy of the remote-file access (RFA) code is spawned so that multiple remote-file operations can occur simultaneously.

5. System performance

Performance is always a difficult thing to measure and can be somewhat subjective. However, some measurements were made and they indicated a remarkable improvement in system response.

Two types of measurements were made, one to measure the unloaded network response and one to measure the response during normal operations. First the unloaded network response will be discussed. To make this measurement, all network traffic was removed and all nonessential tasks were aborted. The extra network traffic had a minor (unmeasurable) effect on response; however, the individual tasks initiating this traffic added some CPU loading for the CPUs involved. These measurements were made with a variety of CPUs and two operating systems, the RSX system and the FALCON stand-alone system. All measurements were made using a master task (packet originator) and a slave

task (ECHO). The master sent a packet with an embedded sequence number, the slave received it and ECHOed it back to the master. The master checked the packet sequence number and sent another; at the end of this process, the average loop time per packet was recorded. All measurements made were for 1000 packets each 64 bytes in length. The results from the various tests are shown in Table 1.

Figure 2 is a plot of three different tests showing throughput versus packet size in bytes. All tests used the master/slave relationship referred to earlier. Curves 1 (11/73s) and 2 (11/23s) illustrate the maximum throughput for two individual nodes using a stand-alone macro program in each. It should be noted that this test is of academic interest only because it is unlikely that any operating system could support this rate. This is really an indication of the system hardware limit.

Curves 3 (11/73s) and 4 (11/23s) were derived using the CPUs indicated on the FALCON system; even this low-overhead system significantly reduces throughput.

Curves 5 (11/73s) and 6 (11/23s) use two FORTRAN codes on the RSX operating system in the master/slave relationship indicated above.

For all tests, it is obvious that for increased packet size the throughput goes up accordingly because there is some amount of fixed overhead in both the hardware (controller) and software. When this overhead is amortized over longer packets, it has a decreasing effect on the throughput. Also, as the operating system complexity increases, the throughput decreases.

Unfortunately, most control systems must operate with short packets, which is the least efficient area on any of the curves. Furthermore, control systems most frequently require a sophisticated operating system to be able to handle the changing demands of the system being controlled. Although the throughput appears small (compared to 10 megabits/s), it is not that small when considering the volume of data that can be transferred even at these reduced rates. For an 11/73 pair for instance, the rate for a 64-byte packet is 140 kilobits/s; for an 11/23, it is 95 kilobits/s.

Measuring system response during normal operation is more difficult because of the random nature of the operation and also because most measurements will slow down the system under test.

First, the system has a "real time" feel during normal operation and does not appear sluggish at all. An attempt was made to measure knob response while the system was unloaded and at full load; the knob response was still rated at 10 updates per second even at full load. Full load was defined as two consoles accessing the same remote node for data to update the graphic displays as well as the knob data itself. In addition, the remote nodes were running their full program load of surveillance, etc.

6. Conclusions

This system has met or exceeded our requirements for speed of response and system flexibility (expandability).

Our control knob update rate of 10 per second under full load is certainly fast enough to give a "real time" feel for accelerator operation. The update rate for the graphics scope, while satisfactory for our present requirements, is really rate limited by the graphics hardware itself and not by network data transfer.

With an ETHERNET system, nodes can be added easily as requirements arise; this capability facilitates adding new functions or splitting off functions from an existing overloaded system.

The ETHERNET system should be easily upgraded to any new bus or ring network architecture if one should prove to be superior in the future.

Acknowledgments

The author would like to acknowledge the efforts of the entire Instrumentation and Control Section of AT-4 (the Systems Integration Group at Los Alamos) for their contributions toward the design and implementation of the original FMIT prototype control system. I would also like to give special acknowledgment to Eric Vanderveer and Tom Zaugg for their efforts in converting the original system application codes to ETHERNET.

References

[1] R.M. Suyama, D.R. Machen, and J.A. Johnson, "The FMIT Facility Control System," IEEE Trans. on Nucl. Sci. 28 (3), 2252 (1981).

[2] "IEEE Standard Modular Instrumentation and Digital Interface System (CAMAC)," American National Standard ANSI Std. 583-1975, IEEE Std. 583-1975, published by The Institute of Electrical and Electonics Engineers, Inc., 345 East 47 Street, New York, N.Y. 10017.

[3] "The Ethernet, A Local Area Network Data Link Layer and Physical Layer Specifications," Compiled by Digital Equipment Corporation, Maynard, MA; Intel Corporation, Santa Clara, CA; and Xerox Corporation, Stamford, CT. Version 1.0, September 30, 1980, Digital Order Number AA-K759A-TK.

[4] E.P. Elkins, "Use of ETHERNET for the FMIT Control System," Los Alamos National Laboratory memorandum AT-4-82:368, December 27, 1982.

Table 1

Unloaded network response

CPU Type Master/Slave	System Type Master/Slave	Loop Time (ms)
11/73 - 11/73	RSX/RSX	7.3
11/60 - 11/73	RSX/RSX	7.6
11/73 - 11/23	RSX/RSX	8.7
11/60 - 11/23	RSX/RSX	8.9
11/23 - 11/23	RSX/RSX	10.8
11/73 - 11/21	RSX/FALCON	6.1
11/60 - 11/21	RSX/FALCON	6.1
11/23 - 11/21	RSX/FALCON	8.8

Figure Captions

Fig. 1. FMIT control-system ETHERNET network.

Fig. 2. ETHERNET performance versus packet size for different operating systems.

Curves 1 (11/73s) and 2 (11/23s) show throughput for stand-alone macro codes.

Curves 3 (11/73s) and 4 (11/23s) show throughput for FALCON/macro system.

Curves 5 (11/73s) and 6 (11/23s) show throughput for RSX/FORTRAN system.

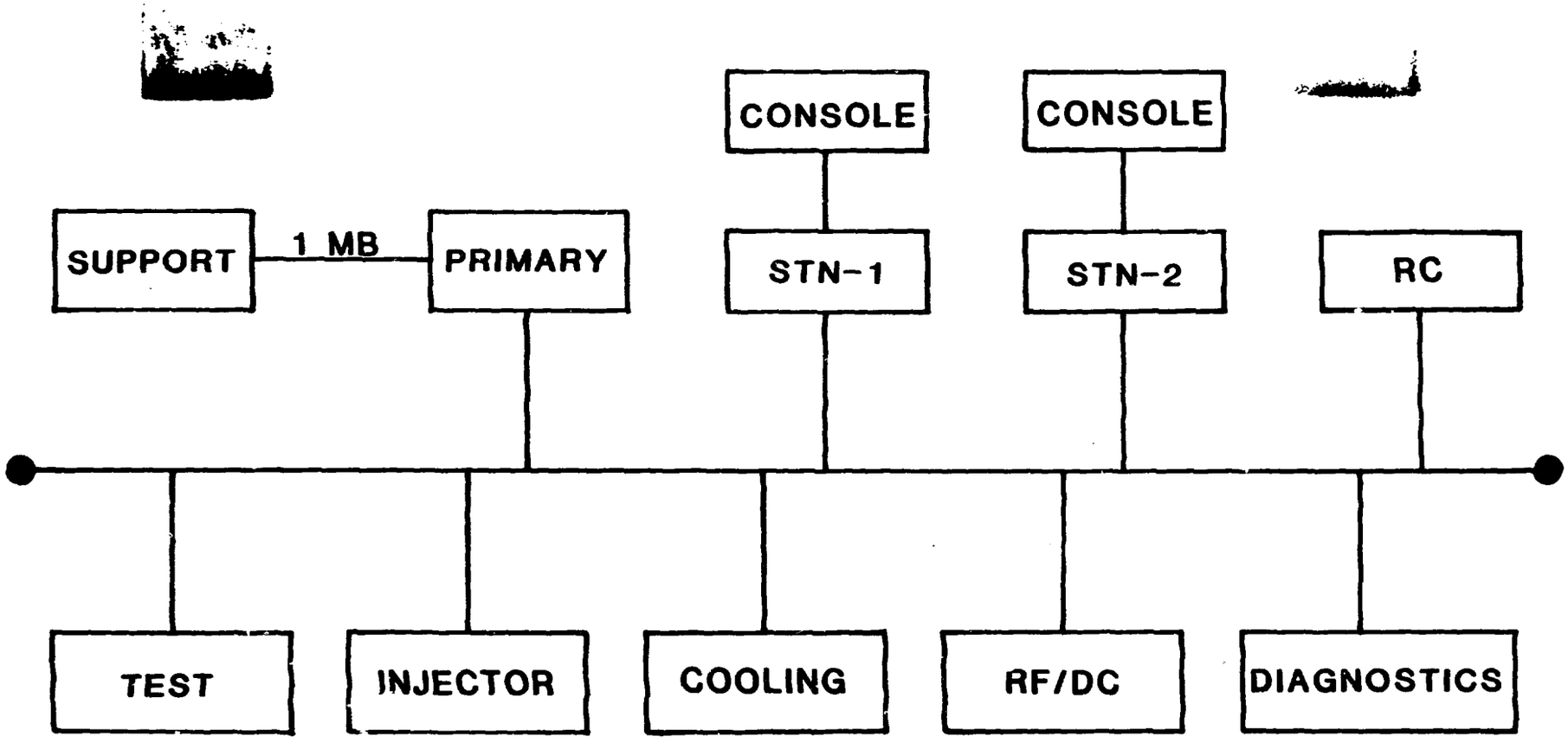
CONSOLE
CONSOLE
SUPPORT
1 MB
PRIMARY
STN-1
STN-2
RC
TEST
INJECTOR
COOLING
RF/DC
DIAGNOSTICS

Fig 1

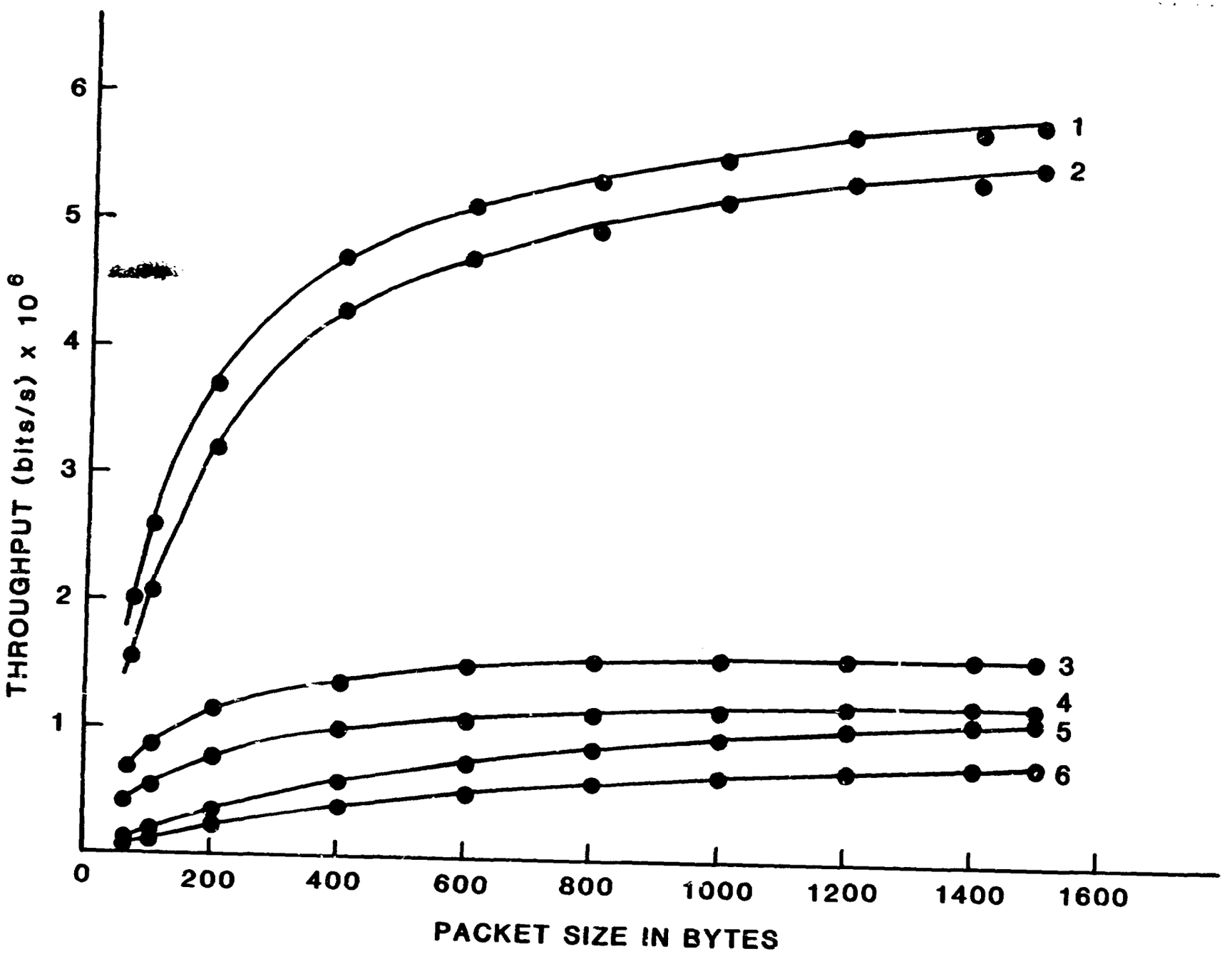
THROUGHPUT (bits/s) x 10^6
6
5
4
3
2
1
0
200
400
600
800
1000
1200
1400
1600
PACKET SIZE IN BYTES
1
2
3
4
5
6

Fig 2