ATLAS Internal Note
DAQ-No-64
28 January 1997

Performance Analysis of ATM Network Interfaces for Data Acquisition Applications

D. Calvet, P. Le Dû, I. Mandjavidze
CEA Saclay, 91191 Gif-sur-Yvette CEDEX, France

M. Costa, J.-P. Dufey, M. Letheren, C. Paillard
CERN, 1211 Geneva 23, Switzerland

Abstract

This presentation addresses some design aspects of ATM network interface nodes for data acquisition applications in the field of High Energy Physics. We present the development of Windows NT and LynxOS device drivers for ATM network adapters. We show a comparative study of the performance of PCI/ATM network adapters on Pentium and PowerPC based platforms. We evaluate the influence of the operating system on performance and measure the overhead of our drivers. We propose several methods to improve the real time characteristics of conventional network interface drivers and show their impact on performance.

I. Introduction

Data acquisition and event selection systems for High Energy Physics experiments, in construction or planned, need to handle larger and larger data volumes in real time. The demand on aggregate bandwidth has grown from a few Mbytes/s for CERN LEP experiments to several tens of Gbytes/s for future experiments at LHC [1], [2], [3]. Efficient communication networks are needed to link several hundreds of detector data sources to a comparable number of processors running the selection algorithms for the on-line filtering of events. The RD-31 collaboration [4] is investigating the use of Asynchronous Transfer Mode technology (ATM) [5] for Trigger/DAQ applications at LHC. One of the goals of the project is to evaluate on small scale demonstrators the feasibility of event builders based on ATM components [6].

In this paper, we investigate the interface between a node (data source or destination processor) and the ATM network. This interface is a key component in the system.

II. ATM Network Based Trigger/DAQ

In High Energy Physics experiments, sophisticated multi-level event selection systems are needed to reduce the raw data flow to a level that can be recorded on permanent storage. The first level of on-line event selection is generally based on fast pipelined logic while subsequent levels use commercial processors whenever possible. We proposed in [7] a model of a Trigger/DAQ system for the second and third level trigger of the ATLAS experiment. The system comprises a number of data sources, destination processors and a supervision unit. An ATM network links all components and is used to carry both data and protocol traffic. The sources buffer event data fragments previously digitized by the read-out electronics of the detector. The destination processors collect event data from the sources and run the on-line event selection algorithms. The supervision unit controls the operation of the system. Messages between various elements are exchanged at a rate up to 100 kHz. The size of messages ranges from less than 100 bytes to several tens of kilobytes. It should be noted that the system relies on quick request/response transactions.

The network of a Trigger/DAQ system has to handle various types of traffic with different needs in terms of bandwidth, priority, routing latency, etc... The ATM technology has been designed to carry simultaneously on a common physical medium various types of traffic having different service requirements. Our simulation studies in [7] showed that ATM nodes and switching fabrics can handle in a single network the various types of traffic specific to Trigger/DAQ applications. These studies were focused on the protocol and data traffic through the network, therefore the simulation model was relatively simple. We did not take into account the software and hardware overheads generated in the ATM network adapters. In particular, the influence of the operating system and device driver in the processor nodes and data sources were ignored. These parameters can be measured on our demonstrators. Then, we can use them to tune the simulation program so that it gives a more accurate picture of the behavior of a real system and allows to predict the performance of larger systems. In this paper, we present the measurement of the most relevant parameters that characterize ATM interface cards.

III. ATM Interface Node Architecture

At present, several tens of vendors offer ATM Network Interface Cards (NIC) for workstations and Single Board Computers (SBC). Cards based on several bus standards are available: VME, SBus, ISA, and PCI bus. Although the original specification of the PCI bus was aimed at the PC/Workstation market, it has also been adopted as a local peripheral bus for mezzanine cards on SBCs [8]. As PCI is gaining acceptance on the market, more and more vendors offer ATM components with a built-in PCI interface [9], [10], [11]. The use of

Presented at the 2nd International DAQ Workshop on Networked Data Acquisition Systems, RCNP, Osaka, 13 - 15 November 1996

PCI/ATM interfaces was therefore a natural choice for our demonstrator. The architecture of a PCI/ATM node is presented in Figure 1.

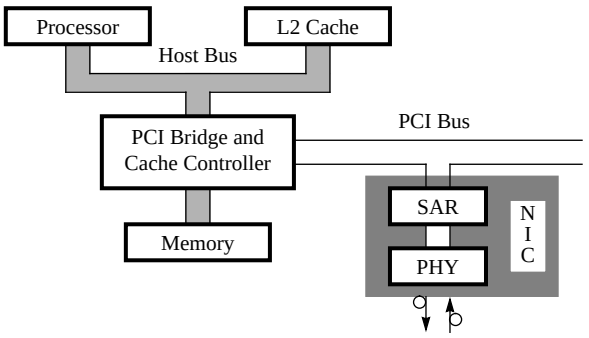

Figure 1: PCI/ATM node architecture.

The host processor, cache and memory are connected to PCI through a PCI bridge. This bridge/memory controller provides a low latency access to the ATM network interface. It also provides a high bandwidth path for the ATM interface to access directly the host memory. The bridge guarantees cache coherency between processor caches and the main memory. The ATM interface includes a Segmentation And Re-assembly engine (SAR), with a PCI master/slave interface. The SAR performs the ATM Adaptation Layer (AAL) and ATM layer protocol functions. The physical layer is handled by the PHY device connected to an electrical/optical converter.

IV. Description Of Platforms Tested

We investigated two different host platforms: a VME single board computer and a PC. The characteristics of these platforms are presented in Table 1.

Table 1.
Characteristics of the platforms used

Platform	Vendor	CPU	Clock	OS
RTPC 8165	CES	PowerPC 604	96 MHz	LynxOS 2.3
PC XMT 5133	DELL	Pentium	133 MHz	WindowsNT 4.0

Both platforms have 32 Mbytes of memory. The secondary level cache is 256 Kbytes on the PC and 512 Kbytes on the VME board. The ATM interface cards [12], [13], that plug in the PC and the VME SBC [14] use the NicStar chip from IDT [9]. This chip includes a PCI master/slave interface and a SAR engine that supports AAL3/4 and AAL5. It implements Constant Bit Rate (CBR) and Variable Bit Rate services (VBR) with three levels of priority. The host memory, rather than on-board memory, is used to re-assemble the packets received from the network, store the packets to be transmitted and maintain various control structures. NicStar uses its PCI master capability to access the host memory in order to reduce host CPU utilization. These features permit the design of low cost and efficient ATM adapters.

V. Operation of an ATM NIC

At present, most commercial network adapters implement the TCP/IP suite. Performance measurements of ATM adapters over TCP/IP has been reported in [15]. Our own experience with commercial ATM cards showed us that the real time performance using TCP/IP is far from what is required for demanding Trigger/DAQ applications. For this reason, we decided to develop the device drivers for our ATM cards. We could locate data transfer bottle-necks and find the limitations of our drivers.

We will now describe how a conventional device driver controls a network interface card. When an application wants to send a message across ATM, it performs a “Send” call to the corresponding device driver. The parameters passed are a pointer to a User Transmit Buffer (UTB), the size of the buffer and a connection identifier that specifies the destination of the message. The “Send” function is executed in kernel mode. It copies the UTB to one or several Kernel Transmit Buffers (KTB) allocated from the non-paged memory pool (flow 1 in Figure 2).

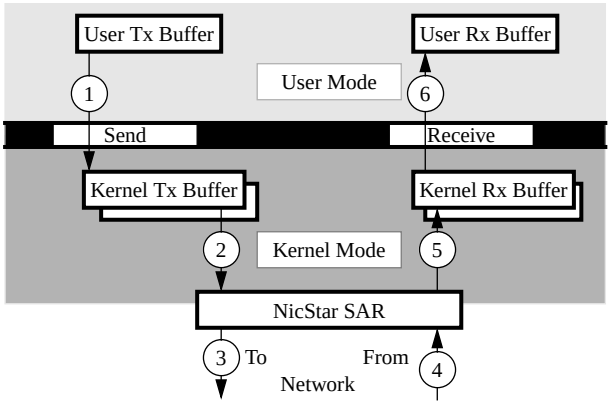

Figure 2: Device driver interaction with NIC.

This copy is needed unless the application can guarantee that the UTB remains resident in the system memory until the buffer has been sent. The “Send” function next places the descriptors of buffers ready for transmission into a queue serviced by NicStar. The “Send” function returns and the rest of the transmit sequence is done by NicStar independently of the host processor. NicStar fetches data from the KTBs in slices of 48 bytes (i.e. one ATM cell payload), forms ATM cells and transmits them via the physical framer (flows 2 and 3).

Before NicStar can receive data across ATM, the device driver must provide it with a supply of host memory locations which may be used for the re-assembly of packets. These Kernel Receive Buffers (KRB) are resident in memory. NicStar reassembles the packets directly in host memory KRBs by demultiplexing the incoming stream of ATM cells on the basis of their connection identifier (flows 4 and 5). On packet completion, NicStar writes KRB descriptors in a queue serviced by the device driver. Optionally, an interrupt can by generated. To read incoming packets, the application performs a “Receive” call to the device driver. It passes a pointer to a User Receive Buffer (URB). The device driver copies data from KRBs to the URB and frees the KRBs for re-use by NicStar (flow 6). The “Receive” function also returns to the application the length of the received packet and a connection identifier that specify its source.

VI. Analysis of Overheads

From the previous description, two types of software overheads can be distinguished. First, the operating system has to switch from user mode to kernel mode for each send or receive transaction. We measured that this overhead amounts to ~10 µs for the platform running LynxOS, and ~60 µs for the Windows NT platform. We observed a larger overhead on the PC because all function calls to a device driver are handled by the I/O Manager, which is part of the Windows NT Executive [16]. This overhead is significantly larger than the transmission time of one ATM cell (~2.7 µs for a 155 Mbit/s link) and degrades the performance for small messages.

The second type of overhead is due to data copies for both transmission and reception. We have measured the speed of memory copy between cacheable buffers that are non-resident in the cache. We found that our PC can perform this operation at ~28 Mbytes/s and the VME SBC at ~36 Mbytes/s. This introduces a performance degradation and places an additional load on the host CPU.

We present in Figure 3 the performance of the drivers we developed using the conventional approach.

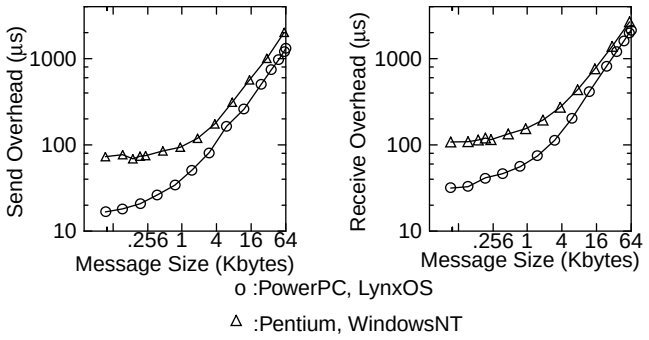

Figure 3: Device drivers' "Send" and "Receive" overheads.

The "Send" overhead is the time for the host processor to complete the "Send" function call. This value does not include data fetch and packet segmentation, both of them being performed by NicStar without the intervention of the CPU. The time needed for segmentation does not add to the host CPU utilization since NicStar and the host CPU run concurrently. The actual time for sending a packet by NicStar is determined by the size of the message and the bandwidth allocated to the corresponding Virtual Channel.

To measure the overhead of the "Receive" function call, an AAL5 packet of a given size is sent to the NIC. After the whole packet has been re-assembled in the host memory by NicStar, the "Receive" function is called. The measurement presented is the amount of time spent by the host processor to pass the received data to an application. Instead of interrupts, the application uses polling to initiate the "Receive" operation.

For both "Send" and "Receive" operations, the overhead due to the user/kernel mode switch and the execution of the function code is dominant for short messages. For large packets, the latency grows linearly. The slope is determined by the speed of memory copy.

VII. Increasing Performance

It can be seen from the previous measurements that the software overheads and memory copy operations are a serious limitation for the efficient use of high speed links. With drivers written as previously described, an application running on the PC can send minimum size AAL5 packets (40 bytes of user data) at a maximum rate of ~15 KHz. This represents a maximum usable bandwidth of ~5 Mbit/s. This is a clear waste of bandwidth for the 155 Mbit/s interface that we investigated. For large packets, a significant additional load is placed on the host CPU due to memory copies. This is a waste of CPU resources. The latency of packet delivery is also increased.

These limitations are well known and several methods have been proposed to make efficient use of available network resources, in particular for small packets [17]. The solution that we describe below uses a similar approach.

First, we mapped NicStar registers and data structures into the application memory space to give the user direct control of the interface. This avoids the penalty of switching between kernel and user modes, because instead of making calls to a device driver, a library of utility functions is called. Second, we modified the buffer management to give NicStar direct access to the User Transmit and Receive Buffers. Therefore, no data copy is needed. The application has to lock these buffers in memory and provide NicStar with the list of physical addresses that describe them. In order to avoid the penalty of retrieving the list of physical addresses from a virtual address for each transmit operation, we suggest that this is done just once when the application program starts. The parameters passed to the "Send" and "Receive" functions are modified. The "Receive" function returns to the user a list of URBs that were used by NicStar to store the received messages. We present in Figure 4 the performance of the optimized library that controls the ATM adapters.

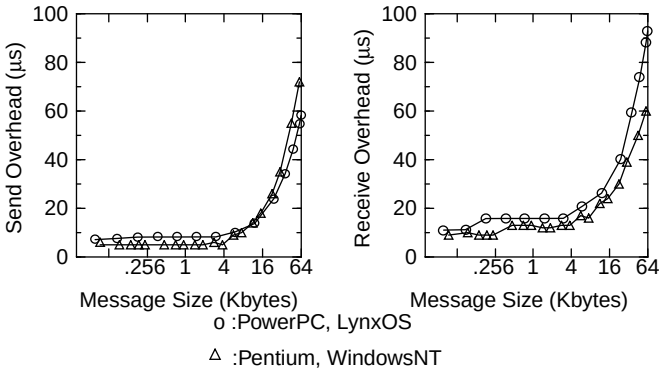

Figure 4: Library "Send" and "Receive" function overheads.

It can be seen that the overheads have been significantly reduced. They do not depend on the speed of memory copy. Overheads increase for packets larger than the size of a physical memory page (4 Kilobytes). For a continuous "Send" and "Receive" operation, we can exploit the full bandwidth of the 155 Mbit/s link for messages larger than ~100 bytes (i.e. 2-3 ATM cells).

VIII. Other Benchmarks

A relevant parameter for the applications that we consider, is the request/response message round-trip latency. We made a test with two identical hosts connected via our ATM interfaces controlled by the optimized library. The master sends a request message to the slave and waits for the reply. We measure the message round-trip latency, T_{RR}. For this benchmark, we set the priority of the application to the maximum value allowed for non-privileged tasks. The distribution of T_{RR} is plotted in Figure 5 for request and response messages of 40 bytes (i.e. one ATM cell).

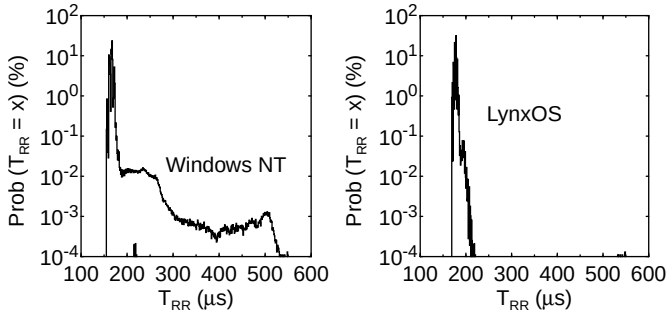

Figure 5: Packet round-trip latency.

The average packet round-trip latency amounts to ~165 µs for Windows NT and ~170 µs for LynxOS platforms. The sum of the CPU overheads for sending and receiving one cell is ~13 µs while the overhead for Nicstar operation and physical transmission is ~70 µs. The sum of these two figures represents half of T_{RR}. Since Windows NT is not a real time operating system, a long tail for the distribution of T_{RR} can be seen. On the other hand, the platform running LynxOS has much better time response characteristics.

We performed the same test with request messages of 40 bytes and response messages of variable size. In this test, the master does not wait for the reply from the slave before it issues the next request (i.e. request messages are pipelined).

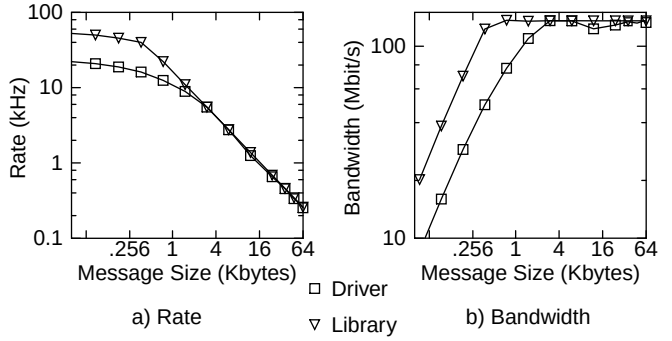

Figure 6: Request/Response benchmark performance.

This benchmark shows the capability of the host and interface to handle traffic in both the transmit and receive direction at the same time. The slave services the requests at a rate that depends on the size of the reply message. The master receives the replies at the same rate. The maximum rate that can be sustained with the VME SBC is plotted in Figure 6.a. The corresponding usable bandwidth is shown in Figure 6.b. For short messages, the system can handle request/response at a maximum rate of ~20 kHz using the driver and ~50 kHz using the library. When the size of messages is increased, the rate is determined by the link bandwidth. The performance on the PCs is comparable. When the application uses the library, saturation of the link (i.e. ~135 Mbit/s) occurs for reply messages larger than 512 bytes.

IX. Discussion

We have presented two types of device drivers for our ATM cards, each one having its own advantages and draw-backs. The conventional approach permits a complete decoupling between the application and the device. This provides security of operation and allows multiple users to share the device transparently. Standard Application Program Interfaces (API) are provided to programmers. However, performance is poor, especially for short messages. This is a serious limitation for the real-time applications that we consider. Commercial network interfaces (ATM, Ethernet and others) use this type of device driver.

With the second approach we propose, it is possible to get a significant increase in performance. The communication latency can be reduced and no additional load is placed on the host CPU for data movement. This is a key advantage for time critical applications. However, there are a number of draw-backs. Because the user has direct access to the interface, the security of operation cannot be guaranteed by the operating system anymore. The user is in charge of the management of shared resources such as the exclusive access to the physical device and to receive/transmit buffers. The use of standard APIs is almost excluded.

At present, the performance of processors running the standard device drivers for NICs is not sufficient to fully exploit the potential of high speed links. This is particularly true for short messages. Although, the power of processors will certainly continue to increase in the future, the limitations that we observe are due also to the access to external devices such as bridges and main memory. During the last decade, the performance of these components did not increase as rapidly as the clock frequency of CPUs. Nevertheless, we can foresee that future host platforms equipped with commercial network adapters will be able to make efficient use of the bandwidth offered by high speed networking technologies.

The approach in the development of efficient device drivers that we have described in this paper leads to satisfactory performance for communications between the host platforms that we have investigated using a 155 Mbit/s ATM link. It is the solution that we can propose today to satisfy the real time constraints of Trigger/DAQ applications.

X. Acknowledgments

The authors wish to thank Creative Electronic Systems SA, especially A. Wiesel for giving us access to technical information and efficient support.

XI. REFERENCES

[1] ATLAS collaboration, “Technical Proposal for a General-Purpose pp Experiment at the Large Hadron Collider at CERN”, CERN/LHCC/94-43, December 1994.

[2] CMS collaboration, “The Compact Muon Solenoid Technical Proposal”, CERN/LHCC/94-39, December 1994.

[3] ALICE Collaboration, “Technical Proposal for A Large Ion Collider Experiment at the CERN LHC”, CERN/LHCC/95-71, December 1995.

[4] M. Costa et al., “NEBULAS - A high performance data-driven event building architecture based on an asynchronous self-routing packet-switching network”, CERN / LHCC/95-14.

[5] L. G. Cuthbert, J-C. Sapanel, ATM - the Broadband Telecommunications Solution, IEE Telecommunications Series 29, ISBN 0 85296 815 9, London, 1993.

[6] M. Costa et al., “Results from an ATM-based Event Builder Demonstrator”, IEEE Trans. on Nuclear Science, vol. 43, No 4, June 1996, pp. 1814-1820.

[7] D. Calvet et al., “A Study of Performance Issues of the ATLAS Event Selection System Based on an ATM Switching Network”, IEEE Transactions on Nuclear Science, Vol. 43. No. 1 February 1996, pp. 90-98.

[8] “Draft Standard Physical and Environmental Layers for PCI Mezzanine Cards: PMC”, Standard IEEE 1386.1.

[9] “IDT 77201 NicStar”, User Manual, IDT Inc., Santa Clara, November 1995.

[10] “SARE PXB 4110”, IC for Communications Data Book, Siemens, August 1995.

[11] “PM 7375 Lazar 155”, Programmer’s guide, PMC-Sierra Inc., March 1996.

[12] “ATM 8468 PCI-ATM Mezzanine Card”, Programming Guide Version 1.0, DOC 8468/PG, Part Number 085. 434, Creative Electronic Systems S.A., May 1996.

[13] D. Calvet, F. Servaz, ”Design of a PMC/ATM Interface”, in Proc. Second International Data Acquisition Workshop on Networked Data Acquisition Systems, RCNP, Osaka, 13-15 November 1996.

[14] “PowerPC Single Board Computer RTPC 8067LK”, Technical manual Version 1.0, DOC 8067LK/UM, Creative Electronic Systems SA Geneva, Switzerland, December 1995.

[15] C. Battista et al., “Permanent Virtual Circuits Configuration and TCP-UDP/IP Performance in a Local ATM Network”,.

[16] H. Custer, “Inside WindowsNT”, Microsoft Press, 1993.

[17] T. von Eicken et al., “U-NET: A user Level Network Interface for Parallel and Distributed Computing”, Proc. of the 15th ACM Symposium on Operating System Principles, Copper Mountain, Colorado, December 3-6 1995.