C 1 5

2nd International Conference on Local Communication Systems: LAN and PBX

PREPRINTS

Universitat de les Illes Balears
Associació de Tècnics d'Informàtica
International Federation for Information Processing

Palma de Mallorca
(Balearic Islands, Spain),
June 26-28, 1991

Preliminary Performance Results for CBN Half-Duplex VME Stations (V1S)

David J. Greaves[*] & Krzysztof Zielinski
Olivetti Research Ltd.
University of Cambridge, Computer Laboratory. Cambridge, England.

February 26, 1991

1 Introduction.

The CBN (Cambridge Backbone Network) offers an ATM LAN/MAN architecture based around a source release slotted ring [1]. The current implementation operates at 512 MHz, interconnecting five sites at Olivetti Research Limited and the University of Cambridge Computer Laboratory.

This paper reports performance measurements of this first implementation in terms of throughput and response time, when using the V1S CBN station interface [2], Motorola MVME 147 68030 CPU cards (20 MHz), the MSNA (multi-service network architecture) fragmentation protocol and the Wanda micro-kernel.

The purpose of this work was to identify throughput limitations in the current station interface and protocol stack, in order to guide future development.

2 V1S Interface.

The V1S CBN station interfaces provide access to the network over the VME bus. These interfaces were designed for minimum complexity, and their raw performance is simply determined by the bandwidth of their buffer RAMs.

All cells waiting to go onto the network or which have just been received share the same RAM array, which consists of four single-ported bytewide CMOS static

[*]Email: djg@cl.cam.ac.uk

RAMs, operating at a fixed rate of 3.2 MHz. Therefore their terminal bandwidth is 102.4 Mb/sec, which is also the peak rate that data will be received off the CBN ring, owing to its multi-channel architecture. Higher bandwidth stations, using greater quatities of RAM, may also be constructed from the CBN chipset, but these are not reported here. As explained in [1], a 512 MHz CBN has four 102.4 Mbit/s data channels. Each V1S stations receives from only one channel, allocated on a per station basis, and always transmits a cell on the appropriate channel for the destination station.

From the functional point of view, the RAM array is organised in one receive FIFO and four transmit FIFOs, one for every channel. Each FIFO is implemented as a circular buffer 256 cells long.

The RAMs are shared by both the receive and transmit sides of the station, and also no word of data can be sent or received without first passing, in FIFO fashion, through them. (For transmit, cells are written into the RAM by the host and read by the ring access chips and vice-versa for receive.) The essential point is that there is contention for the RAM, and the maximum data rate through the stations, when simultaneously transmitting and receiving, is 25 Mb/sec in both directions. The ring itself will support 16 such stations at this rate.

The V1S interfaces perform no protocol processing; cell headers are copied intact over the VME interface, when generated by or for checking by the host processor board. With standard V1S interfaces, there is no DMA, each 32 bit word must be copied by the processor.

The V1S interfaces help the CPU in one way: they allow received cells to accumulate in the interface, in the receive FIFO, until a cell is encountered which generates an interrupt condition. The interface looks in the cell data field at bit fields which are specific to the MSDL segmentation and reassembly protocol (details in the next section). The interrupt condition is programmable on a per VCI (Virtual Circuit Identifier) basis, and can be set to interrupt on:

1. end fragment cell from an MSDL PDU only,
2. beginning MSDL fragment cell only,
3. the both mentioned above,
4. a per cell basis (every cell on that VCI).

No interrupts are required for transmitting; the transmit side is assumed to be always ready, which it is, owing to the light loads on our current network.

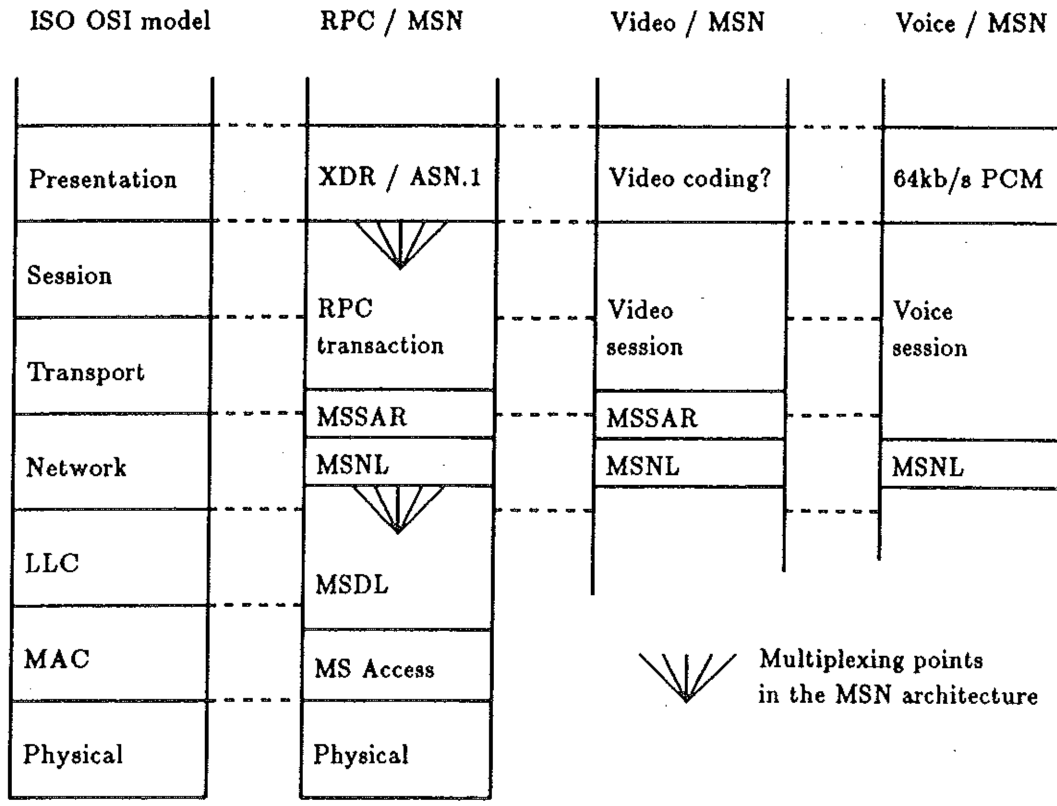

Figure 1: Relationship of MSNA elements to OSI

3 Protocol architecture.

The CBN is currently running under the MS (Multi-Service Network) protocol stack supported by the Wanda micro-kernel. This was originally developed in the Cambridge Computer Laboratory and is currently ported onto many popular workstations and VME boards.

The layered reference model [3] for the MSNA architecture is presented in Figure 1. Certain aspects of this model are similar to those of the ISO OSI reference model. The use of different layers to provide services is still present. However, above the datalink layer, the stacks should be thought of being in parallel depending on the area of application e.g. distributed processing based on RPC, or multi-media applications.

MSSAR (Multi-Service Segmentation and Reassembly) performs block fragmentation into cells and re-assembly at the receiver. The network layer protocol MSNL (Multi-Service Network Level) has no impact on the performance results reported here, since in the MS architecture, network level interconnections are not multiplexed over MS virtual circuits. MSNL simply performs out-of-band connection set up. The link layer protocol MSDL (Multi-Service Data Link) uses virtual cir-

Type	VCI	Data Field	Response
1	16	256+16 = 272	1

Figure 2: Backbone Ring Cell Format. Field sizes are in bits. The response and type bits are supported in the hardware, but not used by MSDL.

VCI	Sequence	RID	Part	Start
16	8	6	1	1

Figure 3: The format of the first 32 bits of a CBN cell when MSDL is being used with block fragmentation. Field sizes are in bits.

cuits, referred to as associations, where there is no hop-by-hop error recovery, and any node involved in the circuit is free to unilaterally terminate it at any time. The MS Access layer contains the device driver and network interface. These low-level protocols have been implemented for several networks, including both fixed and variable packet size networks.

When used over a fixed-size cell network such as the CBN, MSDL uses the physical network cell size. For the CBN, the cell size was chosen to be the same as was used on the CFR (Cambridge Fast Ring) [4]. The design of both networks preceded the CCITT adoption of 5 + 48 bytes [5]. The CBN (and CFR) use a 4 + 32 byte format, with the header including 16 bit VCI and 16 bits of SAR information (Figures 2 and 3).

This paper refers to the CBN V1S interface evaluation, so the main issue is of the efficiency of the MSSAR and MSDL implementation on the current hardware. So that the MS software suite could easily be ported onto multiple different network types and configurations, the subset of functions which have to be implemented in the network driver (MS Access layer) was limited only to the fragmentation operation and the hardware dependent send and receive operations. The more sophisticated re-assembly operation, which is not as hardware dependent, was performed in the separately compiled MSDL layer. Hence on the transmit side, the driver software has direct access to the association's I/O buffers, but the same is not true for the receive side. This approach is fully justified by the demand of easy portability, but in the context of very high speed networking, needs more careful consideration.

The implementation of MSSAR for the Backbone Ring uses the first two data bytes of a cell for fragmentation and re-assembly information. As shown in Figure 3, these bytes fall in the first 32 bit word of the cell, along with the MSDL virtual circuit identifier (VCI). Only this first 32 bit word of the cell needs to be manipulated by the protocol software. The RID is a re-assembly identifier which is common for all cells from one block. The Start Flag is set for the first cell from a block and for this first cell the Sequence number contains the number of cells that compose the block. The Sequence number counts down for each cell so that a value of one indicates the last cell of a fragment. If the Part Full Flag is set, then the cell body does not all contain valid data and the number of valid bytes is stored in the last byte of the cell.

4 Efficiency of MSDL implementation.

4.1 Performance analysis.

The performance results presented in this paper were measured over a CBN of aproximately 10 km long working at a frequency 512 MHz. The rotational latency was 22.7 microsecond and the physical ring contained 7 CBN frames, giving a total of 28 slots. The ring had no other traffic during the experiments. The experiments concerned:

1. Measurement of the maximum transmit operation speed,
2. Analysis of the data transfer speed achieved between two CBN stations,
3. Measurement of the two-way delay transfer time (ping time).

According to the former description, the MSDL block of data is directly copied by the driver software from process I/O buffer space, hence the final speed of the transmit operation is determined by: buffer handling overhead, fragmentation operation overhead, the speed of copying the data to the FIFO in the interface and the underlying performance of the hardware communication subsystem itself. The obtained results have been illustrated in Figure 4. The throughput figure is the rate that was achieved when all of the CPU time was spent performing kernel operations and network IO, with no time spent in application processing.

The increase of transmit speed with data block length is explained by the amortisation of buffer handling cost and the reduction in frequency of interaction with the driver. For the long data blocks, the obtained in-block transmit data rate was over 16 Mbit/s. This corresponds to over 18 Mbit/s actual transmit rate (headers + data). The time of the transmit processing per cell is 15.9 microsecond. This was further analysed using an oscilloscope enabling us to partition this time into: the time of copy from the I/O buffer to the FIFO equal to 12 microsecond (a

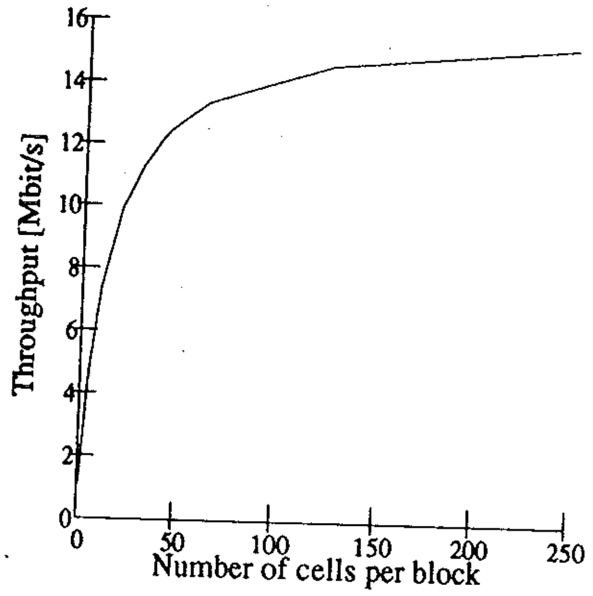

Figure 4: Transmit data rate versus number of cell per block

peak rate of 24 Mbit/s) and the time for MSDL data fragmentation and transmit descriptor modification, equal to 3.9 microsecond.

The first measurements of the data transfer speed between two CBN stations were rather disappointing. A careful review of the receive side software structure made clear that a few improvements were possible:

1. The original MSDL receive software performed the assembly operation of cells into a block of data stored in an association buffer inside the MSDL layer. The data were first copied from the CBN receive FIFO to the buffer inside the CBN driver and next the pointer to that buffer was passed to the association layer as a parameter. Next the data from the driver buffer was copied to the association buffer. Hence the first improvement was removing the additional copy operation.

2. The separation of the re-assembly operation from the driver function required an up call to the appropriate C routine inside the association layer. This caused an additional overhead. So the second improvement was merging of the association handling and re-assembly operations within the driver receive interrupt handler.

3. An analysis of the criticial paths in the combined association handling, re-assembly, and copy algorithms made possible their more efficient coding.

The obtained transfer rate after each step of improvement is illustrated in Figure 5. Stage O corresponds to the original version of software.

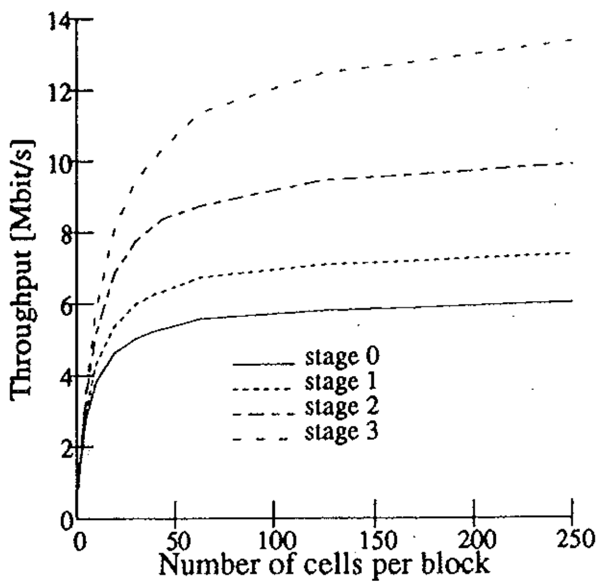

Figure 5: User space to user space data transfer rate (throughput) for different stages of the receive side modification.

For the longest data blocks, the obtained transfer data rate was over 14.2 Mbit/s, which corresponds to over 15.6 Mbit/s transfer rate (headers + data) and was limited by the speed of the receive process. The time of the receive processing per cell is 18.03 μs, which can be split into: the time of the copy from the FIFO to the association buffer equal, as for transmit operation, to 12 μs and the time of the association handling equal to 6.03 μs. So the time of the receive cell operation is only 2 μs longer than the corresponding transmit time.

The two-way delay transfer time (ping time) is the time for a block of data to be sent to a process in the remote station and back to the sending process. This parameter characterises the latency of the communication subsystem and speed of the Wanda kernel. The obtained results are depicted in Figure 6.

These values have been measured by setting the receive VCI interrupt condition for an interrupt on every cell. The particular curves correspond to the different receive side software implementation discussed earlier. The gradient of the obtained characteristic is determined by the per-cell receive processing overhead and the constant overhead corresponds to the Wanda kernel process operation. In the next section, a detailed analysis of the processing overhead and the effect of the interrupt strategy is considered.

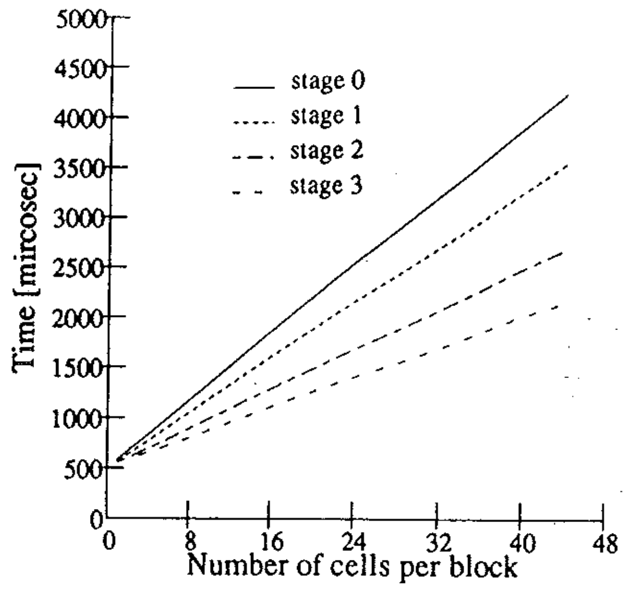

Figure 6: Two-way response time (ping time).

4.2 An analysis of block data transfer processing overhead.

This study addresses the software overhead of data transmission and reception over the CBN using MS and the V1S interfaces. The study technique adopted for estimating the processing overhead at this point is similar to that chosen in [6]. It involves identifying the normal path through the network driver code and counting the instructions.

All of the MSNA protocol software, including the CBN device driver, has been written in C using the GNU GCC68K complier with the optimization option enabled. The results of an analysis of the number of instructions in the compiled code on most common routes along the transmit receive and interrupt handler paths have been summarised in Tables 1, 2 and 3 respectively. The total number of instructions is, in fact, quite small, since the experiments do not include RPC, transport or other higher-level protocols, and because the Wanda IO buffer structure was originally designed for efficient support of the lightweight MSNA datagrams.

On the receive side, the context switching operation and subsequent entry of the receive routine can take up to 42 instructions. Otherwise the receiving of a single cell in a sequence belonging to a common data block needs only 14 instructions overhead. In addition there are the the nine 32 bit copy instructions inevitably required to move the data.

The comparison of the transmit main loop overhead given in Table 1 to the receive loop overhead in Table 2 leads to the unsurprising conclusion that the receive

Operation	No. of instructions
Call and passing parameters	10
Check if broadcast	2
Select channel to send	10
Main send loop repeated for each cell of data block:	
1. Modification of the tran. descriptor and cell header	12
2. Copy data of a cell body to the FIFO	9

Table 1: Transmit block of data overhead

Operation	No. of instructions
Call and passing parameters	13
Look up association	10
Main receive loop repeated for each cell per block:	
1. Checking of a cell header and mod. of an assoc. state	14
2. Copy data from the FIFO to an association buffer	9

Table 2: Receive data routine overhead

loop is slower than the transmit loop (by two instructions). This fact prevents a receiving station from getting further interrupts during the receive processing of the cells belonging to a common data block, even when the interrupt condition is set on a per-cell basis. This would not necessarily be true, however, on a more heavily loaded network, or when the cells have been bridged over multiple (heterogeneous) ATM networks. The disparity in transmit and receive rate is absorbed in the interface receive-side FIFO buffers.

As has been described, in the revised implementation of the software, the transmission and reception of the data block is performed completely by the CBN driver software. Once buffers have been allocated, the only overhead which depends on the Wanda kernel implementation is its context switching time. The Wanda kernel provides a multi-threaded environment optimised for real-time application software and networking. Nevertheless, the pure software implementation of context switching represents a considerable host overhead when compared to a dedicated host processor solution, or to a machine which context swaps in hardware, such as the Transputer.

For the CBN/V1S receive side, the degree of context swapping is mainly influenced by the incoming data rate and the interrupt strategy selected for the incoming VCI. The dependence of the two-way response time on interrupt startegy has been illustrated in Figure 7. The graph shows that the response time is almost

Operation	No. of instructions
Call and passing parameters	3
Interrupt entry	4
Interrupt exit:	
1. Going back to int. routine	7
2. Descheduling the thread	2
3. Save the thread state	12
Total	28

Table 3: Receive data interrupt service overhead

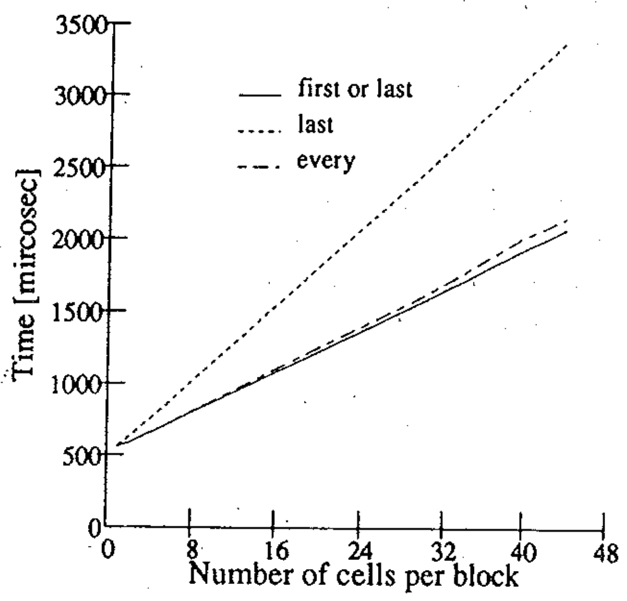

Figure 7: Two-way time delay for different interrupt condition settings.

independent of whether the interrupt condition is set for start of block or on every cell. In contrast, setting the interrupt condition for the end of data block breaks the receive-transmit process pipelining and leads to a substantial increase of the latency.

We can usefully identify three types of network communication:

1. where response time is critical,
2. where throughput is critical,
3. where the cell arrival rate is relatively low.

For VCIs which fall into case 1, interrupting on every cell is appropriate. For VCIs in case 3, interruption only on the last cell of a block may be appropriate, provided we can be sure that the interface receive FIFOs are not likely to overflow (of which there is currently no warning). Evidently there is a trade off between context switching overhead and increase of latency caused by the beaking of the transmit/receive pipeline and the consequent loss of cut-through.

In the single processor CBN station, owing to the currently lightly loaded rings and homogeneous stations (i.e. they were all the same), we have seen that the trivialised solution to this trade-off can be achieved by proper tuning of the transmission and reception speed. Because the transmit routine is slightly faster than receive, even when set for interrupt on the every cell basis, there was no degradation of the performance caused by the context switching, since the receive interrupt handler encountered back-to-back cells from the interface. However, we must also consider broader networking scenarios, including multiple active streams, non-trivially loaded networks, heterogeneous processors with different clock speeds, and the impact of bridging to other networks and ATM switches. In order to then minimise context switching, while preserving the ATM benefit of cut-through, the use of anticipation in the form of dally delay loops in the receive handler should be considered. In some cases however, dally loops may not be desirable, owing to the waste of otherwise useful processor cycles.

As has been stated in the previous section, and as is clear from the graphs, the length of the data block as a logical unit should be as large as can reasonably be supported by the application.

5 CBN versus Ethernet over MSNA

Since several of our CBN stations are also fitted with Ethernet connections, a direct comparison between the two under the same MSNA protocol stack was an

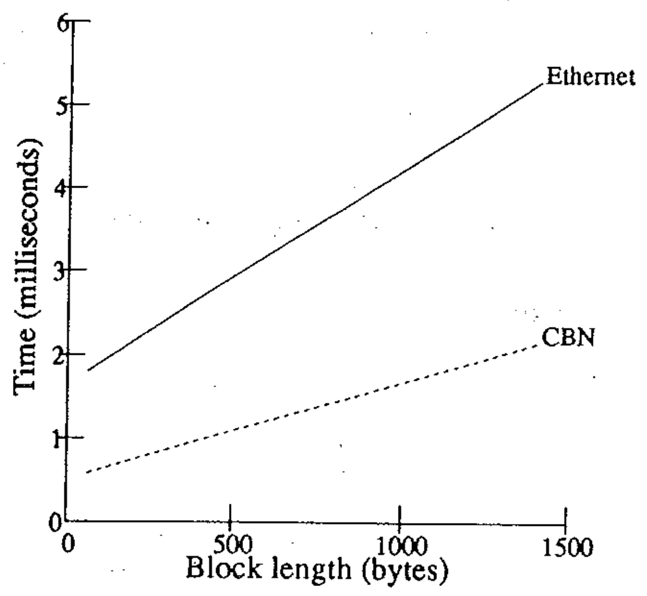

Figure 8: Ethernet and CBN two-way delay over MSNL comparison.

obvious experiment. The standard Wanda Ethernet driver was used, but apart from this, the software used for each type of network was identical.

As a basis for the comparison study, the two-way time delay (ping) was chosen. The results of experiment have been depicted in Figure 8. The ping operation over CBN takes 1.2 millisecond less than over the Ethernet using a short packet and 3.1 millisecond less using a large packet of data. This effect is mainly explained by the virtual cut-through ability of the CBN ATM type communication and a much lower starting time of the CBN network interface.

From the functional level, the ping operation represents the simplest version of RPC communication. This is the case of a process-to-process communication when marshalling and unmarshalling procedures are empty and the remote process returns a result requiring only trivial processing.

6 ANSA RPC performance evaluation

As a contrast to the simple ping, a full function RPC system from the ANSA project (Advanced Networked Systems Architecture) was tried. Two tests were performed on each of the two networks. The *echo* RPC test returns as a result the data block that was received as a parameter. In contrast the *sink* RPC result ignores its parameters and its result is always null. Therefore, the primary difference is that in the echo test, the data block is transmitted twice.

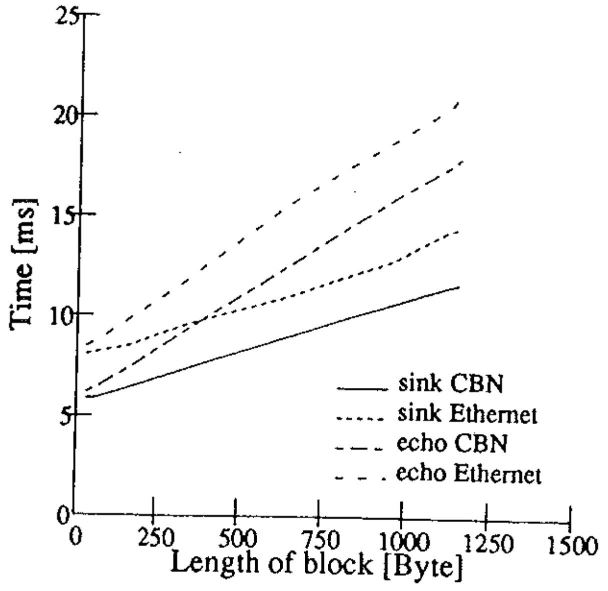

Figure 9: ANSA RPC time for CBN and ethernet.

The results for both tests are shown in Figure 9. These show that ANSA RPC over the Ethernet is always between 2 and 3 millisecond slower than the CBN. Although this is consistent with the results obtained using MSNA ping, it is of reduced benefit owing the minimum of 5 milliseconds ANSA protocol processing time. This overhead results from the ANSA implementation, which was designed for maximum portability, rather than speed.

7 Direct Memory Access.

A slightly enhanced CBN interface, capabable of performing DMA as bus master, over the VME bus, was constructed and tested. It copies one cell at a time to an address written into the interface by the VME host processor. The copy is commenced by the writing of this address into the interface and the exact operation (send, receive etc.) is specified by the address that the address is written to. Completion of the copy is not signalled by the DMA interface, but it is assumed to be quick, and in any case, the VME bus is normally locked while the copy is taking place. A bus-error is generated, should the copy not be complete when another is requested. Therefore, no explicit synchronisation is needed. The first 32 bit word of the cell is not copied by the DMA unit since it contains protocol information. Instead, the processor reads or writes this to the appropriate FIFO before initiating the copying of the data.

Using the DMA interface, the maximum MSDL throughput was increased to

24 Mbit/s (27 Mbit/s including headers). This compares favourably with the results without DMA, shown in Figure 4 where a maximum rate of 15.23 Mbit/s was achieved. Since only one DMA interface was constructed, only transmit speed could be measured.

Two benefits from the use of DMA were expected. First, there is potential for parallelism, since data can be copied while the CPU is processing the received or yet to be transmitted data and protocol information. Second, the data copy itself should be faster, owing to the hardware implementation. Unfortunately, neither of these potential benefits fully materialised. This can be explained by the realtively low bandwidth available over the VME bus into or out of the 'local' RAM on the CPU board. The delay from VME DS to DTACK experienced by the DMA bus master was nearly one microsecond. This gave an overall cell copy time of 7.4 μs (c.f. 12 μs for software implementation). The protocol processing time remained at 3.3 μs. The bus arbitration time before the start of each cell transfer was measured as 300 ns. Since the data rate is limited by the copying time, rather than the protocol processing time, it is reasonable to assume that the maximum receiving rate would be the same as the measured transmit rate.

8 Future Work and Conclusion.

On the measurement side, we need to perform similar experiments on a more heavily loaded network. The V1S interfaces include false traffic generators, so that the load can be artificially increased. We can also use other network stations as traffic generators. Another interesting case is the duplex communication situation, where the interference between simultaneous receive and transmit processing needs examining.

As far as hardware development goes, we must bear in mind that the V1S interfaces were designed as a cheap interface with about 30 Mbit/s simultaneous receive and transmit throughput. In practice, there has been an impact on host throughput as a result of the current CBNs being operated at only about half their design speed, that is 512 MHz instead of 1000 MHz. This has increased the contention resolution time for FIFO buffer access and the 12 microsecond FIFO copy time might have been expected to be about 6 microseconds in a full speed system.

Apart from DMA, future interfaces will also include optional demultiplexing of interleaved received streams on separate VCIs, although this would not show to advantage in the tests reported here, where only a single active VCI was used. We are also building CBN stations with more advanced buffer architectures, to meet demanding applications, such as an enhanced multi-media file server.

In general, it seems that the only fully satisfactionary interface solution requires a dual-processor architecture, where the 'host' processor is relieved of as much context switching overhead as possible. With MSNA, it is possible to envisage relatively simple protocol hardware which can examine data structures generated by the kernel's IO subsystem and provide for MSDL header generation, checking and DMA address calculation. However, such hardware is protocol specific and can easily go out of date. The results in this paper enable us to predict that a current technology, general purpose processor might be able to keep up with per-cell processing for rates up to about 50 Mbit/s (75 Mbit/s with 48 byte cells). Such a general purpose processor is inevitably more flexible and programmer friendly. It is also able to 'mop up' occasionally encountered error cases which would probably be left to the host in a hardware solution. In either case, the host CPU loading will be very much reduced. It is important to point out that data should still be copied directly to/from the CBN FIFO's to the process I/O buffers as in the currently used single processor architecture.

Acknowledgements.

Many thanks for invaluable help are due to Joe Dixon, Mark Hayter and the rest of the Wanda group at the Computer Laboratory. Thanks are due to the rest of the systems group, and especially to D (Mac) McAuley for the MSNA picture. At Olivetti Research, Dimitris Lioupis, David Milway and Andy Hopper greatly contributed to the Backbone Ring project. The work in this paper was funded by Olivetti Research Limited.

[1] 'The Cambridge backbone network.' DJ Greaves D Lioupis and A Hopper. IEEE Infocom 90, San Fransisco June 1990. Also in Proceedings European Fibre Optic Conference (EFOC / LAN 88) Amsterdam, June 1988.

[2] 'Backbone Ring V1S station interface specification.' DJ Greaves. Olivetti Research Limited. 1990.

[3] 'Protocol Design For High Speed Networks.' University of Cambridge technical report 186. DR McAuley. December 1989.

[4] 'The Cambridge Fast Ring Networking System.' A Hopper and RM Needham. IEEE transactions on computers, Vol 37 no 10. October 1988.

[5] CCITT I series recommendations, especially I.321, Geneva May 1990.

[6] 'An analysis of TCP processing overhead.' DD Clark, V Jacobson, J Romkey and H Salwen. IEEE Communications Magazine, June 1989.