Preliminary Performance Evaluation of SandiaXTP on ATM at ORL*

Glenford Mapp

Email: gmapp@cam-orl.co.uk

Olivetti Research Limited
Old Addenbrooke's Site
24a Trumpington Street
Cambridge
CB2 1QA
UK.

Abstract

With the deployment of high speed networks based on ATM, multimedia systems are quickly becoming an integral part of distributed computing environments. Protocols which can transport multimedia data are essential for the success of this new enterprise. XTP is one such protocol. In this paper we examine the issues involved in porting SandiaXTP to the ORL's ATM environment and assess its performance using some preliminary tests.

Keywords: ATM, XTP, SandiaXTP

Motivation

Increasing processing power and faster networks will soon make it possible to build large multimedia environments at a reasonable cost. Asynchronous Transfer Mode (ATM) techniques are being used to meet the temporal demands involved in the transfer of multimedia data, including interactive audio and video.

Olivetti Research Limited (ORL) has a long tradition of research into multimedia systems using high speed networks. Pandora [**Hopper90**], a multimedia platform developed in the late Eighties, explored the creation of a multimedia environment using a networked-attached peripheral called a Pandora's Box. The networking for this environment was based on the Cambridge Fast Ring [**Hopper88**].

Recently, ORL has developed a new network architecture based on the "exploded workstation model" in which multimedia peripherals such as cameras and audio devices are attached directly to the high speed network [**Wray94**]. We have constructed a number of such devices along with an in-house 100 Mbits/sec ATM network. These devices include an *Audio Brick* for handling audio, a *Video Brick* which is essentially an ATM camera, a *Video Tile* which is a network framestore with an LCD display and a *Disk Brick* which is a storage system built using RAID technology. See Figure 1.

The hardware is based on an ATM interface card called an **ATMos** board. The ATMos board essentially consists of an ARM processor, 8 MBytes of memory, an ATM network interface and an interface known as the ATOMIC interface. Special cards are added to the ATMos board via the ATOMIC interface to make a network device. These devices run a lightweight, message-based operating system called **ATMos** which provides a modular software environment to build modules which control and manage them [**French95**].

Several multimedia applications with different transport requirements have been developed for this environment. Interactive applications such as video-conferencing can use unreliable connections in which lost or corrupt frames are tolerated in order to maintain low latency. However, for sophisticated applications such as face and voice recognition [**Samaria93**], [**Walker95**], which examine multimedia data looking for certain characteristics, the reliability of the data is more important than the latency of the connection. In certain situations it is better

*Appeared in PROMS '95, Second Workshop on Protocols for Multimedia Systems, "Mozart on Multimedia Highways" ,Salzburg, Austria. October 9-12, 1995

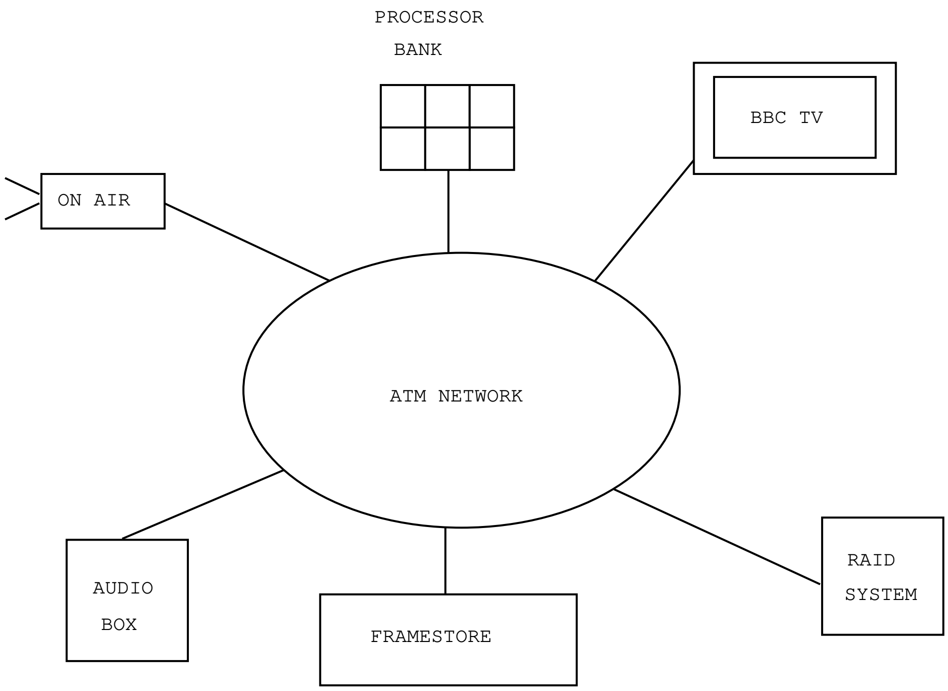

Figure 1: ATM Devices

that there is no retransmission as the next new item of data, e.g. a video frame, will be transmitted shortly and retransmitting old data only introduces jitter.

These different transport requirements must be simultaneously satisfied by these devices. In a conventional computing environment, these different **quality-of-service** or qos requirements would need different transport protocols, e.g. TCP and UDP. Using one transport protocol which can satisfy a wide spectrum of qos requirements may represent a better approach. The **Xpress Transfer Protocol** or XTP [**Strayer92**] can provide such functionality. XTP is managed by an organisation called the XTP Forum and a public domain version called **SandiaXTP** is being developed at Sandia National Laboratories in Livermore, California. In this paper, the port of SandiaXTP to the ORL ATM environment is described and preliminary performance evaluation results are given.

XTP: A brief overview

The current specification, XTP 4.0, is being implemented by several members of the XTP Forum. A more detailed view of the protocol is given in [**XTP Forum95**].

A connection is identified by a 64-bit key and 64-bit sequence numbers are used to represent bytes in transit. XTP does not define an addressing scheme of its own but supports several commonly used addressing formats including IP. The system also uses the IP checksum algorithm but does not use a pseudo header in the checksum calculation. There is a **sort** field in the XTP header which indicates the priority of a given connection. This allows low latency data, such as audio, to be processed before other less demanding media. Multicast connections are also supported.

XTP has other fields to specify quality of service requirements. There are mechanisms to indicate whether user data should be checksummed and whether corrupt or lost messages should be retransmitted. These may be specified on a per packet basis. The frequency of acknowledgements is under the user's control and both *go-back-n* and *selective repeat* retransmission techniques are supported.

Rate-based and windowing flow control mechanisms are employed on each connection. The window size can be configured by the user at the start of a connection. There are *traffic* parameters with which the user can specify several qos requirements including the maximum size of data segments that can be transmitted as well as the inward and outward average and burst rates. For ATM networks that provide qos guarantees, it may be possible to map XTP traffic parameters directly onto the underlying network qos parameters.

SandiaXTP

SandiaXTP is implemented using C++. The code is divided into two sections. The first is the **Meta-Transport Library** or MTL which contains generic protocol functions such as FIFO management routines, checksumming as well as transmitting and receiving packets from the underlying data service. There is an abstract data delivery class from which classes for running over real networks are derived. The code supports data delivery classes for running over UDP and raw IP. A more detailed layout of the code is described in [**Strayer94**].

The SandiaXTP section of the code implements the XTP protocol. The protocol is run in a separate user space process known as the XTP daemon or **xtpd**. Each XTP connection is represented by an instance of a context class and one context manager class manages all the XTP connections on a given host. SandiaXTP does not currently support streams of different priorities or multicast connections.

Users interact with xtpd via a shared message queue built on top of the local IPC mechanism. There are also shared buffers between xtpd and the user for receiving and sending data. When the user wishes to send data, it is first copied into the shared send buffer before it is sent on the network. The user interface supports a number of operations including configuring, registering and binding a context, after which the user can connect to other destinations or listen for new connections. The configuration call allows the user to set several qos parameters on a connection including the window size, the frequency of acknowledgements and the retransmission policy. There are separate calls to get and set the traffic parameters.

Packet management is done using an XTPpacket class which is derived from a generic packet class in the MTL Library. In SandiaXTP there is a global packet queue which is used to send and receive data. Sending data onto the network is a synchronous operation and the packet is returned to the packet pool when sending is finished.

ORL's ATM environment

ORL began building its ATM network in late 1992 and so predates some of the ATM standards developed by the ATM Forum. In particular, the system uses a local meta-signalling mechanism, developed at the Computer Laboratory, University of Cambridge [**McAuley90**], to setup and manage ATM connections. There are plans to move to the ATM Forum signalling standards in the near future.

As well as directly-attached devices, DEC Alpha workstations currently running OSF 2.0 are also connected to the ATM network. These workstations have locally developed network interface cards (NICs) of which there are two versions. The older **Yes** NIC card, provides the basic service of transmission and reception of ATM cells. All per cell handling, including segmentation and reassembly, has to be done by the workstation. Recently, a new NIC card called the **TCAT** card has been developed which off loads these ATM protocol functions onto an on-board ARM processor. Presently, both cards are in active service.

The ATM switches at ORL are also based on the ATMos card. There are 4-port and 8-port switches. The 4-port switches operate at 200,000 cells/sec while the 8-port version switches cells at 1Mcells/sec. Topology determination as well as support for mobile devices are being explored using these switches.

The switches as well as ATM peripherals run the ATMos operating system. This is a single address space operating system, so an image may contain several processes. Currently, there is no support for dynamically starting a new process. User processes may have one of two priorities. ATMos uses message passing as its basic communication paradigm and sending a message to a process of higher priority will result in the current process being suspended and the higher priority process being scheduled. There is a native threads package but the **POSIX threads** package is also supported. ATMos supports the concept of an **active queue** where the processing of ATMos messages is executed in the sender's context with interrupts disabled so there is no context switch involved.

Porting SandiaXTP

Porting SandiaXTP to the ORL ATM environment can be divided into two major tasks. The first is porting the ATM environment to the SandiaXTP code on the 64-bit Alphas running OSF and the second is porting

SandiaXTP, which was written for a UNIX environment, to ATMos. In both cases, we wanted to keep the code needed to be changed to a minimum. We detail some problems below.

Porting the ATM environment to SandiaXTP

This basically involved implementing a new data delivery service for ATM. The generic data delivery service defined in the MTL library proved insufficient to support a connection-oriented service such as ATM. So we added some calls to do so. We believe that there should be two generic network classes one for connectionless systems and the other for connection-oriented networks.

It was decided to use one ATM connection between two hosts. We are aware that this allows the multiplexing of different streams on the same ATM connection [**Tennenhouse90**] but since the ATM network does not presently support the relevant qos parameters and SandiaXTP is yet to support stream priorities, there was little to be gained by insisting on a one-to-one mapping between a context and ATM connections. This can be readily modified once the necessary functionality is present to make use of it.

As previously stated, the SandiaXTP code assumes a totally connectionless model of operation. The data delivery service is given the packet as well as the full destination address. Since we did not want to change the semantics of the send operation, it was necessary to dynamically map a given destination address to an ATM connection in the ATM data delivery service. Performance was improved by caching the last ATM connection on which data was sent. The effect of this was that SandiaXTP was completely unaware of the ATM connections being made on its behalf. The advantage of this approach is that it was possible to timeout and remove ATM connections when they were unused for a given length of time. Connections were remade as required resulting in only active connections were supported by the ATM layer.

Porting SandiaXTP to ATMos

The port of SandiaXTP to ATMos presented some major challenges. Firstly, the shared message queue between the XTP daemon and the user was implemented as discrete, synchronous ATMos messages where a user would send and immediately wait for a reply. We implemented routines which resembled the shared memory system on Unix. This was not too difficult since ATMos is a single address system, where pointers can be passed freely. Some functionality associated with Unix is not provided under ATMos including signals, getting a user_id, finding out if a client process had terminated, etc.

The XTP daemon in ATMos continually runs in a loop waiting for ATMos messages from various sources, including the network, which indicates incoming messages for remote connections, the timer and local clients. This structure prohibits the use of blocking in other parts of the system for any length of time hence the ATM data delivery service in ATMos had to be written in an asynchronous manner to handle different events.

The only exception to this rule is when the XTP daemon attempts to send to a destination for the first time. Since SandiaXTP assumes a synchronous send, the connection to the destination host must be made synchronously. This is fine once the connection can be made without any trouble. If not all the active streams will be affected.

Improving the code

SandiaXTP uses a dual copy architecture in which data is copied first to the shared send buffer. There is a further copy in the UNIX network driver which is given pointers to the header and data. In ATMos, this second copy must be done in the ATM network class for XTP since the ATM driver does not support scatter-gather operations. This requirement allowed us to implement a fast checksum-and-copy routine in assembler which improved the speed of the code.

The other major improvement was using an active queue to recycle receive buffers without a context switch. When the ATM networking code receives a buffer for SandiaXTP, it executes the function for a receive ATMos message on the active queue of the XTP daemon. This function takes the receive buffer and places it in a local packet receive queue and then attaches an empty buffer to the receive message which is used by the ATM driver to get more data. Hence no context switches are involved in getting data from the network but an additional message must be sent to notify the XTP daemon that something has been received from the network. This mechanism operates well on a heavily loaded network since several receive packets will be in the local packet

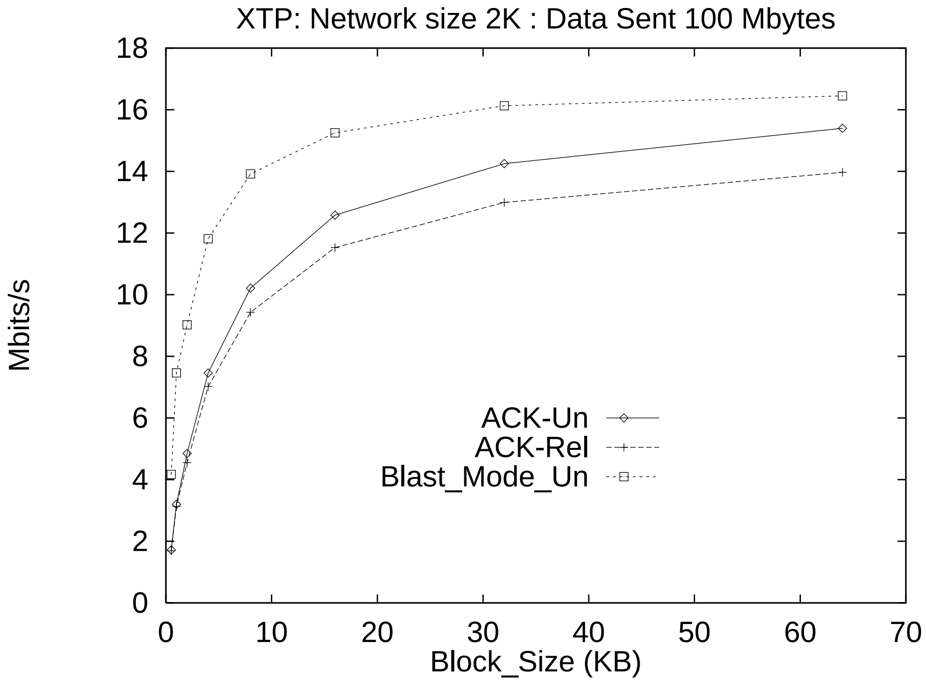

Figure 2: Throughput vs Send Buffer Size

queue before the XTP daemon actually runs.

Testing

Our main interest was to find out how fast the system could perform in two modes of operations. The first is synchronously; i.e., the sender sends a buffer and blocks until it is acknowledged by the remote XTP daemon whereupon the sender continues sending. There are two context switches involved in sending data: one from the sender to the XTP daemon and the other from the XTP daemon back to the sender. It was interesting to keep the total number of bytes transmitted the same but the size of the buffer that the sender gives the XTP daemon was varied from 512 bytes to 64 Kbytes. We were also interested in measuring the cost of reliability so the above process was done twice: the first with totally reliable send semantics and the other with totally unreliable send semantics. In the results, these are denoted by ACK-Rel and ACK-Un respectively.

The other mode that was thought to be interesting is what is called the **blast mode**. Again the total amount of data is kept the same but the data is sent in short bursts with a defined wait period between bursts. The sender does not wait for acknowledgements. Data is sent using totally unreliable semantics. In the results, this is denoted by Blast_Mode_Un.

It should be stressed that these experiments were geared towards measuring maximum throughput. Hence the results obtained were in the absence of any detectable networks errors which required retransmission for reliable XTP connections. In addition, window sizes were set to twice the send buffer size in the synchronous mode and much higher in the blast mode to avoid any delays due to a small window size.

We also wanted to see the effect of changing the network buffer size. So two network buffer sizes were chosen: 2Kbytes and 4Kbytes. The total amount of data sent on each occasion was 100 Mbytes. In burst mode, this was divided into bursts of 1 Mbytes long and a waiting period of 100 milliseconds between bursts. Two parameters, throughput and CPU usage, were measured as the send buffer size was varied. The throughput is measured from one end to another by the applications. The results are given for ATM peripherals running ATMos.

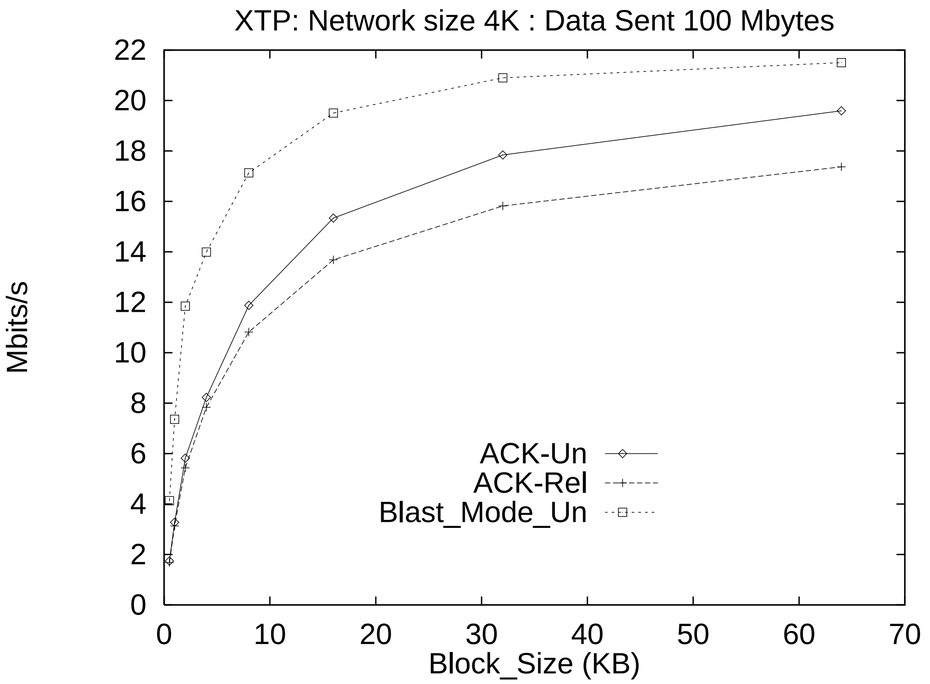

Figure 3: Throughput vs Send Buffer Size

Results

Figure 2 shows the throughput as the send buffer size is increased using a network size of 2Kbytes. There is a wide variation in throughput. There is low throughput for small packets in the synchronous mode. This indicates that there may be a lot of overhead involved in communicating between the XTP daemon and the user. This overhead involves the two context switches, which were mentioned earlier, an ATMos message being sent and the reply obtained, finding the relevant context as well as formatting and decoding the commands for the XTP daemon to execute.

For a given block size, the difference between the ACK-Un and the ACK-Rel represents the cost of reliability in terms of reduced throughput. This cost appears to increase as the send buffer size increases. When data is being sent unreliably, it is simply copied into the network buffer. When data is being sent reliably, the fast checksum-and-copy routine is employed. The former is quicker than the latter. However, the point to note is that the difference at the network buffer size is important since for bigger send buffers the data must be segmented into the network buffers so for large send buffers the difference at a given size will be proportional to the number of network buffers that are required to be sent.

In addition, for a given send block size, the difference between ACK-Un and Blast_Mode_Un can be used as an indirect measure the end-to-end latency as it represents the cost, in terms of throughput, of the need to wait for acknowledgements from the other side before proceeding. This measure was about 2.45 Mbits/sec for 512-byte blocks and between 1Kbyte and 4Kbyte blocks was fairly constant at around 4.3 Mbits/s, dropping steady afterwards to 1.05 Mbits/s for 64Kbyte blocks. This indicates that the end-to-end latency is fairly high for small blocks when compared to large blocks. This may be due to the fact that large send blocks will benefit more from the improved ability to receive a large number of network packets without doing a context switch.

Figure 3 shows the results for the same tests but this time 4K network buffer sizes was used. The graph shows that there is some improvement in throughput but the shapes are similar. For 2Kbyte network buffers, the maximum throughput for ACK-Rel was 13.97 Mbits/s, for ACK-Un it was 15.4 Mbits/s and for Blast_Mode_Un it was 16.45 Mbits/s. With 4Kbyte buffers the figures were 17.37 Mbits/s, 19.59 Mbits/s and 21.5 Mbits/s respectively. There was no increase in the throughput for small packets.

Figure 4 shows the amount of CPU usage measured at the transmitter and the receiver for reliable and unreliable synchronous transmission. In blast mode, instantaneous CPU usage proved too difficult to measure accurately. For the transmitter, CPU usage climbs fairly gradually to total CPU usage. For the receiver however, the

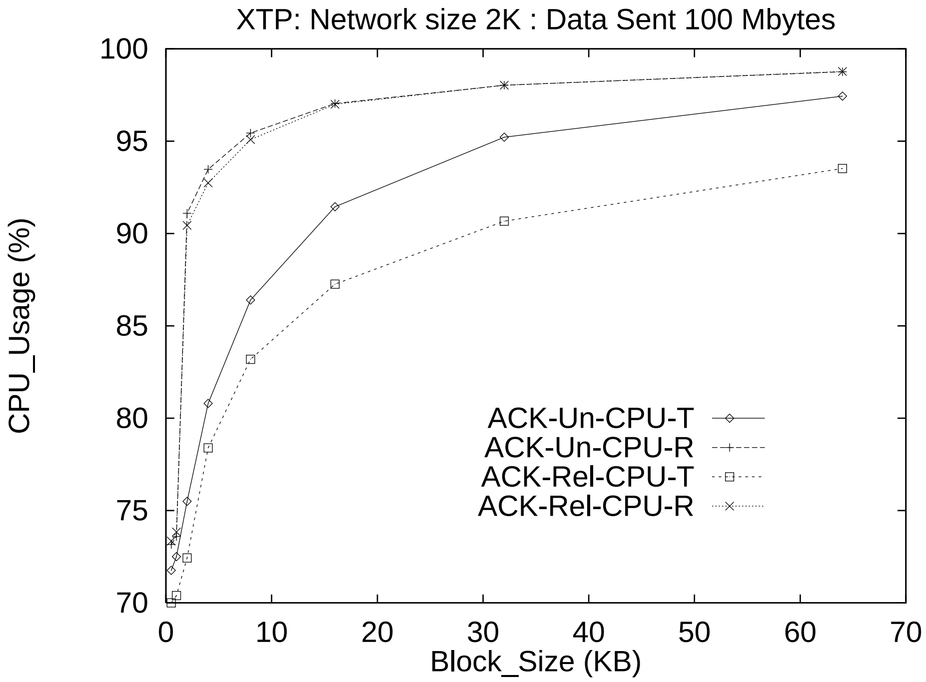

Figure 4: CPU Usage vs Send Buffer Size using 2KB Network Buffers

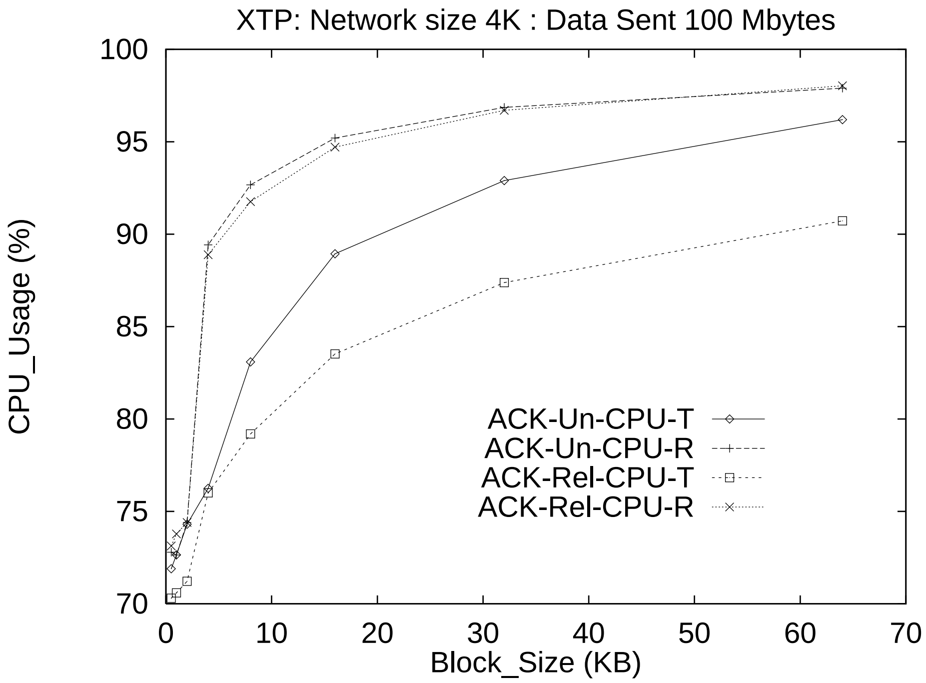

Figure 5: CPU Usage vs Send Buffer Size using 4KB Network Buffers

movement towards total CPU usage is different. In particular, there appears to be a sharp increase in CPU usage when the send buffer size becomes equal to the network buffer size. This occurs for both reliable and unreliable transmission. We have no firm explanation for this and investigations are continuing. Figure 5 shows the results for 4Kbyte network buffer sizes, the results are similar although the CPU usage is generally lower. This apparent mismatch between transmitters and receivers highlights the need to have proper qos support such as those in XTP to prevent receivers being swamped by one or more transmitters.

Conclusions

Some of the observations above are due to how the XTP protocol is implemented in SandiaXTP rather than the protocol itself. For large blocks, good throughput is achieved, while low throughput and high latency have been found for small blocks. In terms of multimedia, it is necessary to address this issue since audio uses relatively small blocks and has very strict temporal requirements. The results show that SandiaXTP would only support a small number of high quality audio channels.

There has been some discussion about doing a kernel version of the SandiaXTP code for Unix systems. We believe that this will be useful and would probably increase performance. However, in the ATMos environment, where the core system is made up of a set of processes rather than a monolithic entity, we believe that there may be greater benefit in trying to run most or all of the XTP protocol in the process space of the application. This can be done by using threads and implementing the protocol as a user space library. We would also like to explore issues of Application Layer Framing (ALF) and Integrated Layer Processing (ILP) [**Clark90**] in this context. We are presently examining these issues.

Acknowledgements

Special thanks to Martin Brown of ORL for writing the low level ATMos device drivers for the network devices, Gray Girling and Ian Wilson for their work on ATMos and Tim Strayer for his hard work on SandiaXTP.

References

[**Clark90**] D.D. Clark and D. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of ACM SIGCOMM '90*, pages 200–208, September 1990.

[**French95**] L.J. French, I.D. Wilson, and D.P Gilmurray. *ATMos III System Manual.* Olivetti Research Limited, 1995. Edited by C.G. Girling.

[**Hopper88**] A. Hopper and R Needham. The Cambridge Fast Ring Networking System. *IEEE Transactions on Computers*, 37(10), October 1988.

[**Hopper90**] A. Hopper. Pandora – an experimental system for multimedia applications. *ACM Operating System Review*, April 1990.

[**McAuley90**] D.R. McAuley. Protocol Design for high speed networks. Technical report, Computer Laboratory, University of Cambridge, January 1990.

[**Samaria93**] F. Samaria and F. Fallside. Automated Face Identification Using Hidden Markov Models. In *International Conference on Advanced Mechatronics.* The Japan Society of Mechanical Engineers, 1993.

[**Strayer92**] W.T. Strayer, B.J. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol.* Addison and Wesley, 1992.

[**Strayer94**] T. Strayer, G. Simon, and R. E. Cline Jr. An Object-Oriented Implementation of the Xpress Transfer Protocol. *XTP Forum Research Affiliate Annual Report*, pages 53–66, 1994.

[**Tennenhouse90**] D. Tennenhouse. Layered Multiplexing Considered Harmful. In *Protocols for High-Speed Networks.* Elsevier Publishers, BV, 1990.

[**Walker95**] R. Walker. An Application Framework for Speech Recognition in Medusa. Technical report, University of Cambridge, 1995. Computer Science Tripos Pt II.

[**Wray94**] S. Wray, T. Glauert, and A. Hopper. The Medusa Applications Environment. In *International Conference on Multimedia Computing and Systems*, May 1994.

[**XTP Forum95**] XTP Forum. *Xpress Transport Protocol Specification.* XTP Forum, March 1995. Rev 4.0.