SPECIAL ISSUE — HIGH PERFORMANCE PROTOCOL ARCHITECTURES

41 Guest Editorial

42 Some Lessons Learned from Various ALF and ILP Applications
   A GHOSH, J CROWCROFT

48 ALF/ILP Based Automated Implementation of Distributed Applications
   T BRAUN, I CHRISMENT, C DIOT, F GAGNON, L GAUTIER

55 A Comparison of Automatic Protocol Generation Techniques
   R DE SILVA, L DAIRANE, A SENEVIRATNE, M FRY

61 Some Issues for Dynamic Synthesis of ALF/ILP Systems
   M FRY, A GHOSH

66 Towards Predictable ILP Performance — Controlling Communication Buffer Cache Effects
   B AHLGREN, M BJÖRKMAN, P GUNNINGBERG

SPECIAL FEATURES

71 Errata

72 book Reviews

Published for Australian Computer Society Incorporated
Print Post Approved PP255003/01660
CONTRIBUTIONS

All material for publication should be sent to:
Mr David Wilson
School of Computing Sciences
University of Technology, Sydney
PO Box 123, Broadway NSW 2007, Australia

Prospective authors may wish to consult manuscript preparation guidelines published in the February 1995 issue. The paragraphs below briefly summarise the essential details.

Types of Material: Four regular categories of material are published: Papers, Short Communications, Letters to the Editor and Book Reviews.

Generally speaking, a paper will discuss significant new results of computing research and development, or provide a comprehensive summary of existing computing knowledge with the aim of broadening the outlook of Journal readers, or describe important computing experience or insight.

Short Communications are concise discussions of computing research or application. A letter to the Editor will briefly comment on material previously appearing in the Journal, or discuss a computing topic of current interest. The term 'computing' is interpreted broadly to include descriptions of computer hardware and software, and major computer applications such as information systems, expert systems, computer networks and office automation.

Refereeing: Papers and Short Communications are accepted if recommended by anonymous referees. Letters are published at the discretion of the Editor, and Book Reviews are written at the Editor's invitation upon receipt of review copies of published books. All accepted contributions may be subject to minor modifications to ensure uniformity of style. Referees may suggest major revisions to be performed by the author.

Proofs and Reprints: Page proofs of Papers and Short Communications are sent to the authors for correction prior to publication. Fifty copies of reprints will be supplied to authors without charge. Reprints of individual papers may be purchased from Associated Business Publications Pty Ltd, PO Box 267, Umina, NSW 2257. Microfilm reprints are available from University Microfilms International, Ann Arbor/London.

Format: Papers, Short Communications and Book Reviews should be typed in double spacing on A4 size paper, with 2.5 cm margins on all four sides. The original, plus two carbon copies, should be submitted. References should be cited in standard journal form, and generally diagrams should be ink drawn on tracing paper or board with stencil or Letraset lettering.

Ideally, a hard copy, a 3½" Apple disk in QuarkXpress or Pagemaker, alternatively Microsoft Word is required. Straight text papers may be submitted in ASCII on 3½" disk. Papers and Short Communications should have a brief Abstract, Key word list and CR categories on the leading page, with authors' affiliation as a footnote. The authors of an accepted paper will be asked to supply a brief biographical note for publication with the paper.

Copyright © 1996 Australian Computer Society Inc.

Production Management:
Associated Business Publications Pty Ltd
(ACN 002 528 154), PO Box 267, Umina 2257.
Telephone (043) 41 3628. Fax (043) 42 5624.

All advertising enquiries should be referred to the above address.

Subscription/Change of Address Form

Send all correspondence regarding subscriptions to the Circulation Manager, PO Box 319, Darlinghurst, NSW 2010 Australia.
Guest Editorial

The growth of high speed networks has prompted the development of distributed applications that have demanding performance requirements. Applications that use multiple media types are prominent examples.

The end-to-end performance of distributed applications is influenced by many factors. Performance is an issue at all levels of the protocol stack. While the speed of the interconnection network is obviously important, in recent years attention has also turned to upper layer, end-to-end protocols. These protocols are realised within the end systems of communication. In a high speed network the upper layers are now often the performance bottleneck.

Research has shown that end-to-end performance is determined by complex interactions, involving the set of protocol functions, protocol implementation techniques, and support services offered by the operating system. Significant performance improvements have been achieved in research laboratories by fine tuning existing implementations or by designing special purpose communication systems. These special purpose systems are often application specific, taking advantage of communication patterns and communication requirements of a particular application. While we now have a much better understanding of communication system behaviour, the construction of such systems is still a difficult and time-consuming task. The challenge now is to provide tools that will permit the building of high performance systems.

Clark and Tennenhouse from MIT have suggested the principles of Application Level Framing (ALF) and Integrated Layer Processing (ILP). ALF is an architectural principle which gives the designer greater flexibility to exploit application behaviour. A significant consequence of ALF is the permeation of the natural Application Data Unit (ADU) throughout the communication system. This data unit is completely application specific, e.g. a screen of information, a talk spurt, an image tile, etc. Clark and Tennenhouse have demonstrated evidence that performance is significantly improved if the ADU can be the unit of processing at all protocol layers. ILP is an engineering principle that reduces the costs of manipulating and copying data. The memory subsystem can be a significant bandwidth limiter. ILP attempts to reduce the number of memory accesses, and also to improve locality of reference for greater cache benefit.

The High Performance Protocol Architectures (HIPPARC H) project is developing and exploring the ALF/ILP principles. In particular, HIPPARC H aims to develop a set of tools that will automate the construction of communication systems from modular components. The project is a collaboration between researchers from the Institut National de Recherche en Informatique et en Automatique (INRIA) in Sophia Antipolis, University College London (UCL), the Swedish Institute for Computer Science (SICS) and University of Technology, Sydney (UTS). In 1994/5 the project was funded by the EU to promote European/Australian collaboration. It has now received further funding under the ESPRIT program. The project has included several work packages, many involving collaboration between the partners. As part of the project two international workshops have been held. Papers were submitted to these workshops from a wide spectrum of the high speed networking community. The first workshop was held in Sophia Antipolis in December 1994.

This special issue contains a selection of papers that were presented at the 2nd HIPPARC H Workshop in Sydney in December 1995. All papers were refereed by an international Program Committee. The papers included in this special issue were also chosen by the Program Committee for publication. Papers were chosen to represent work undertaken by the various partners in the project. Some papers represent collaboration between partners. The papers also derive from different work packages, and thus illustrate for the reader different aspects of the problem of high performance protocol architectures.

The paper by Ghosh and Crowcroft from UCL introduces readers to the need for Application Level Framing. This is done by reporting on the construction of three distributed applications using the ALF principle. The paper illustrates the benefits of constructing communications systems based on application requirements.

Braun et al from INRIA describe the development of a compiler that generates communication system stubs. The compiler takes a formal specification in ESTEREL that identifies the application’s requirements, and generates a system that conforms to the ALF principle. There are a number of different approaches to configurable communication systems. One important issue is static vs dynamic configuration. De Silva et al discuss and analyse some different approaches.

The paper by Fry and Ghosh explores the synthesis of systems from modular components that adhere to the principle of ILP. Certain trade-offs are identified between modularity, flexibility and performance. Finally, Ahlgren et al from Sweden investigate cache management issues for ILP-based communication. Evidence is provided as to the effectiveness of different strategies.

Together these papers introduce the reader to some of the critical issues and approaches to high performance protocol architectures. For space reasons most papers have been condensed from their original length. The authors would be happy to supply the full workshop papers to interested readers. We hope that these papers will enhance the general understanding of the reader, and perhaps motivate some to investigate these issues further.

Associate Professor Michael Fry  
School of Computing Sciences  
University of Technology, Sydney  

Associate Professor Aruna Seneviratne  
School of Electrical Engineering  
University of Technology, Sydney
Some Lessons
Learned from Various ALF and ILP Applications

A Ghosh
J Crowcroft
Department of Computer Science,
University College London,
Gower Street, London WC1E 6BT, UK
{atanujon}@cs.ucl.ac.uk

Application Level Framing ALF is considered a mechanism for building high bandwidth applications. We have built three diverse applications to show that even for low bandwidth applications it has its merits. A problem however is that a good knowledge of both the application and networking is required in order to build ALF based applications. There are currently no aids in the building of ALF based applications. We attempt to identify some common components in our diverse applications which could be used to aid in the building of future ALF based applications.

1 INTRODUCTION

With traditional network applications the reliability, integrity and ordering of the data is provided by using standard communication stacks. So typically an application does not need to be concerned about the mechanism by which data is actually transferred. A file transfer application will have no knowledge of the underlying networks, it simply opens a connection to a remote host, once a connection is established a reliable pipe will be available for use by the application over which files can be transferred. From the point of view of an application the network can be viewed as a "black box" and communication stacks are provided to enable data exchange. This fits in well with our computer science model that we hide as much detail as possible and provide simple interfaces to complex systems. An application is not required to know about the intricacies of packet sizes, loss, corruption and many other details which are managed by communication stacks.

Hiding the communication details simplifies applications. However, some flexibility is also lost, the applications requirements may not map well onto the functionality provided by the communication stack. A file transfer application may well be able to accept, blocks from a file in any order (as long as they all arrive), but there is typically no mechanism for signalling this option. This is a common problem which often occurs when hiding a complex system behind a simple interface fine grained control is lost. Different applications have different requirements, so ideally they should be able to express these requirements to the communication stack. Unfortunately, there is often a mismatch between an applications requirements and what is offered by a communications stack (Wakeman et al, 1991).

As we move towards higher speed networks the overheads imposed by conventional communication stacks start to become too high. A proposed solution is using Application Level Framing ALF (Clark and Tennenhouse, 1990).

The communication stack is greatly reduced and delivers packets to an application with little or no processing. The communication stack does little more than demultiplex a packet to the correct application. Once an Application Data Unit ADU, is delivered to an application it should be processed totally to completion. The theory is that an ADU is structured by an application such that it can be processed to completion immediately upon arrival. This contrasts with the way a communication stack, may be forced to delay before handing a packet to an application due to loss or reordering. If a packet is not processed immediately it arrives then valuable processing time is lost.

A communication stack may perform integrity checks on the data which could be better performed by the application, in order to avoid the penalties incurred by memory bottlenecks. Integrated Layer Processing ILP (Abbott and Peterson, 1993; Gunningberg et al, 1991; Braun and Diot, 1995) concepts could be used to integrate integrity checks with other processing of the data, in order to take advantage of the locality of reference.
One we move to using ALF concepts applications have to become network conscious, so there is a shift from the "black box" view of the network (Christment, 1994).

Features of networking such as loss, reordering and delay are no longer hidden.

Each ALF style application is potentially put in the position of needing to re-invent the wheel with respect to its networking interactions. In this paper we attempt to identify common components of communication systems, which could be provided to ease the building of a future generation of applications. Many lessons have been learned from the implementation of the current generation of transport protocols such as TCP and the ISO protocols (Braden, 1989; Watson et al, 1987; Clark et al, 1989; Colella et al, 1985). It is important that these lessons should not be lost.

A small set of diverse applications have been built to try and exploit ALF and ILP concepts. All the applications use the User Datagram Protocol UDP. UDP as the name implies is an unreliable datagram protocol; it provides little more than packet routing and a rudimentary checksum.

The manipulation of the data in the packets and any extra protocol processing is performed by the application. One of the applications vid (Ghosh et al, 1994) is a video decoder which uses the Real Time Protocol RTP (Schulzrinne et al, 1995). The RTP header processing and the decompression of the data is all performed by the application.

The applications:

- **vid** — A simple video decoder
  A simple h.261 (ITU 1993) video decoder which decodes the RTP format.

- **slogin** — A simple/secure login program
  A simple secure login protocol which exploits the characteristics of the limited speed at which an individual is able to type.

- **mmd** — A multicast mail delivery system
  A multicast mail delivery system which simplifies the administration and delivery of mail to mailing lists.

Traditional applications that have required reliability have been built using reliable protocols such as TCP. For applications that require total reliability and for applications that can tolerate loss, duplication and reordering of packets, it is possible to hand craft protocols that uses UDP and are more efficient in terms of number of packets sent and are more "responsive" to the user.

### 2 VID

The video decoder uses UDP datagrams containing a RTP payload. Loss of some packets in a stream does not cause the video to be unduly delayed. Some experiments at UCL sending video over TCP have shown that due to the requirement for retransmission if a packet is lost there is a tendency for the end to end delay to build up. Any loss of packets will tend to add at least one round trip time (RTT) of delay to the playout of the video. As more losses occur the playout delay will tend to increase. In a video playout application although reliability is desirable some packet loss can be tolerated. Some loss is more tolerable to a huge build up in the end to end delay. A negative feature of TCP from the video playout point of view is that packets containing video may actually be in the TCP input buffers, but if packets ahead of the packets in the input buffer are missing then the packets will not be made available to the application. It is possible to envisage a video application which does solicits retransmissions but will provide the packets to the application as soon as they arrive so that the heavy computation involved in video decompression can be performed immediately a packet arrives rather than allowing a queue of packets to develop which require processing. Another negative aspect of using TCP for video delivery is that TCP does not support multicast i.e. multipoint delivery.

### 3 SLOGIN

There are two popular remote login programs in use on the Internet telnet (Postel and Reynolds, 1983) and rlogin (Kantor, 1991): both these protocols use TCP. It is important in an application such as a remote login program that packets are reliably delivered, so it would seem that TCP is the ideal choice for such an application. If one considers a remote login application at one end of the connection is a user who is typing at the other end output is generated from the commands typed as well as possibly the echoing of the typed commands. A user is not able to type data very fast compared to the data rates possible from networks. A user can only read at a low rate compared to the potential rate available from a network; of course a word processing application may need to send several kilobytes occasionally to update a display. The user is interested in low latency not high data throughput.

In an attempt to reduce the number of packets exchanged, hence keep the load on the network to a minimum, TCP implementations use two mechanisms: the delayed ack and the Nagle algorithm (Nagle, 1984; Braden, 1989). The goal is to reduce the number of packets exchanged by increasing the amount of information carried in each packet; effectively this is achieved by delaying the transmission of packets to allow more information to be carried in each packet.

For a long delay path using TCP decreases the responsiveness of the connection for remote login programs, this is a well known side affect of the delayed ack and the Nagle algorithm. The delayed ack typically does not increase the latency but does prevent the sending of ack only packets.

The slogin program is relatively simple: it uses the UDP layer and uses its own packet format. Various ideas from TCP implementations are used such as delayed acks and the mechanism for calculating the RTT. The slogin program will generate more packets than an equivalent remote login program using TCP because the Nagle algorithm is not used. The Nagle algorithm is in TCP not just for remote login applications but for any application which generates a few bytes at a time which can be amortised into larger packets. The Nagle algorithm seems inappropriate for remote login sessions over long delay paths. The slogin protocol sends packets immediately;
characters are generated by the user in order to keep the response time to a minimum.

4 MMD

The Multicast Mail Delivery MMD, uses multicast to achieve the delivery of the same mail to multiple recipients. There exist many large mailing lists which have recipients around the world. A mailing list is administered at one site. In order to send mail to a mailing list firstly the mail is sent to the administering site where a list of all the members of the list is held. Once a piece of mail reaches the administering site a unicast connection is made to all the members of the list and the mail is individually delivered to each site. For a large mailing list of 1,000 members the administering site will need to make 1,000 individual unicast connections.

The MMD protocol uses the experimental multicast backbone MBONE (Casner, 1993) to distribute mail to multiple recipients, hence reducing the load on the administering site. A much smaller number of packets are exchanged internationally in the best case for a small message 500 bytes only one multicast data packet needs to be sent. Local delivery at each site is achieved by using SMTP over TCP, this is acceptable as the main hurdle of distributing the mail from the administering site to the recipient sites has been overcome. The delivery of mail should be reliable whether the mail is going to a single recipient or multiple recipients. So the MMD protocol must and does support reliable delivery.

The protocol uses two channels one for control information and one for data. The administering site sends periodic status messages, these messages carry a range of the messages available as well as a time interval. When a site receives a status message and discovers that a message is being advertised that it does not have it can send a request for this message. There is potentially a problem; if a new mail message is available and there are 1,000 potential recipients then they all may send a request at roughly the same time, this is the ack implosion problem. In order to guard against the ack implosion problem all control and data messages are multicast so that; any site which has a message can respond (giving some level of redundancy), once a request for a message is made other sites will not repeat the request or repeat the actual transmission of the message. There is still a problem with messages being synchronised, in order to guard against this the status message contains a time value, each receiver chooses a random interval within this range and only when this time is reached can any packets be sent. If the administering site detects that there has been an ack implosion then the time interval can be increased as the group of receivers grows. It is not actually necessary to wait for an ack implosion to occur before increasing the time interval, once a threshold of responses to a status message is reached the time interval can be increased.

A deliberate decision was made not to send out mail messages in an unsolicited manner so that the number of requests for a each message could be used to estimate the size of the group of recipients.

5 LESSONS

We have built three diverse applications; from our knowledge of the underlying networks we are able to build optimal applications. For the three applications described the focus of our optimisation has differed. In the case of vid most of the effort was expended in trying to increase the performance of the decompression stage. For slogin we attempted to decrease the latency for long delay links. For MMD we attempted to reduce the load on administering site as well as reducing the time for a mail message to be delivered.

The applications considered are fairly diverse but a set of characteristics have been identified. We present in table 1 which characteristics were observed in particular applications.

Table 1: Application Characteristics

<table>
<thead>
<tr>
<th></th>
<th>vid</th>
<th>slogin</th>
<th>MMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable Delivery</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ordered Delivery</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Fragmentation</td>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Concatenation</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Compression</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Multiplexing</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Flow Control</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Rate Control</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Congestion Control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error Control</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Unicast</td>
<td></td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Multicast</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1 Reliable Delivery

All the data must eventually be delivered. This implies duplex communications with that exchange of acknowledgments as well as a retransmission scheme to cater for lost packets. Vid for example uses the RTP, if a packet does not arrive at the receiver then no attempt is made to solicit a retransmission. With both slogin and MMD it is essential that all the data eventually arrives at the receivers. A missing block in a mail message or losing characters typed into a slogin connection would not be acceptable behaviour. If reliability is required then the data must be stored at the sender for possible retransmission.

5.2 Ordered Delivery

The data must be delivered in the same order that it left the sender. There may be missing packets, hence no implication regarding reliability. Networks will occasionally reorder packets; for the slogin application it is essential that the characters typed are used in the order they were typed. For the MMD application the order of arrival of the packets is not important as they can be stored as they arrive (as well as being available...
for retransmission), however as reliability is also required it is expected that eventually all the data will arrive, at which point a whole message can be presented to the mail system. For the vid application which is playing out video frames a frame later than a frame already displayed would give an incorrect representation of the video stream.

5.3 Fragmentation
In an ideal ALF world there would be no need for fragmentation. However in the world of real networks there will be instances where an ADU is larger than the MTU. This problem occurs with the vid program when receiving h.261 video, some frames take up to three packets. In this case it is necessary to delay processing until all the fragments are present, breaking the ALF concepts of processing immediately a packet arrives.

5.4 Concatenation
Situations can arise when an ADU size is so much smaller than the MTU that many ADUs may be transmitted together. None of the applications considered required concatenation.

5.5 Compression
A mechanism for supporting high bandwidth applications is to compress the data prior to transmission. It is common for video applications to compress the video before transmission. The vid application receives video compressed with the h.261 standard.

5.6 Encryption
With the growth and widespread use of networks such as the Internet, end to end security is becoming a serious issue. The slogin program can encrypt all traffic so that passwords and other secure information cannot be compromised by wire tapping at intermediate nodes.

5.7 Multiplexing
When a packet enters a host various fields in the packet are used to make a decision as to which process should receive this packet. These fields are used to demultiplex the packet. In the instance of an ethernet packet carrying an IP packet which in turns carries a UDP packet there are several demultiplexing points three such points. Each demultiplexing point incurs overhead. In the case of UDP as no real protocol processing is performed, it should be possible to have a single demultiplex point allowing an incoming packet to go directly to the application.

5.8 Flow Control
A mechanism to reduce or totally stop the amount of data flowing across a connection. If one end of a connection has run out of buffer space and can therefore not accept any more data, then in a protocol supporting flow control it will be possible to signal the peer that there is no longer any buffer space.

In TCP the flow control is provided by every packet sent carrying the number of bytes (window) that the sender is able to receive.

The slogin protocol uses an implicit flow control mechanism. With slogin no information regarding buffer space is sent to the peer, however, the flow control is achieved by never allowing more than one unacknowledged packet. So unlike TCP only one packet can be sent at a time.

5.9 Rate Control
Flow control is used to advertise how much buffer space is available at the receiver. No account has been taken of the rate at which packets can be received, if packets are sent at too high a rate then it is possible buffer over run may occur at the receiving host. In order to guard against buffer over run a rate control mechanism was proposed in VMTP (Cheriton, 1988). In VMTP an interpacket gap time is used.

In the MMD protocol rate control is used to prevent over run at the receiving sites as well as being network conscious by not periodically imposing heavy loads.

5.10 Congestion Control
Networks will occasionally become congested.

Typically once a possible congestion condition has been detected (usually signalled by packet loss) the transmitter sends at a lower rate. Such a scheme is used in TCP (Braden, 1989) to avoid congestion.

It is important that any tailor built protocols have inbuilt congestion control, otherwise the new generation of protocols will drive the networks into congestion collapse.

5.11 Error Control
In order to detect any packet corruption during transit through the network. Packets have some checksum or crc computed across all the data in a packet. The checksum or crc is carried in the packet if on packet reception the checksum or crc does not match the newly computed checksum or crc across the data the packet is considered to have been corrupted in transit and hence discarded.

All the applications considered used UDP to transport their packets. UDP provides an optional checksum which was enabled in all cases. The UDP checksum of the packets were performed before the packets were delivered to the applications so there was no scope to integrate the checksum processing into the ILP loops attempted in the applications.

5.12 Unicast
Until fairly recently it has only been necessary to consider point to point communications. Communications between two peers.

5.13 Multicast
Unlike unicast communications multicast is one to many or many to many communication.

5.14 Redundancy
Generally redundancy is carrying extra data in order to counteract the effects of packet loss.
The slogin protocol attempts to achieve low latency by carrying all unacknowledged data in all packets sent. This ensures that there is not a whole delay RTT if a packet is lost.

5.15 Interoperability
The ability to interoperate with an existing protocol. The MMD and slogin protocols are new protocols so there were no interoperability issues. The vid program was required to use a pre-defined packet formats (Schulzrinne et al, 1995; Turletti and Huitema, 1995), in order to interoperate with existing "standards".

6 APPLICATION COMPONENTS
From the application characteristics and the knowledge gained from building the applications some areas have been identified where additional functionality would aid with the construction of a new generation of applications. Three main areas have been identified:

1 API additions
A set of additions to the API that provide an ALF application finer control and knowledge of the underlying network.

2 Control Functions
Common control functions, an example would be calculating retransmission timeouts. A set of functions which are commonly used in building communications protocols.

3 Data Functions
Some common manipulation functions on the data have been identified. Although being able to identify and provide a common set of data functions is important, more crucial is to determine the interface to the data functions. Ideally the data functions and their interfaces should be chosen such that a pipeline of data functions can be integrated to form a integrated loop, giving ILP.

7 API ADDITIONS
Certain additions could be made to the API in order to aid the building of a future generation of applications.

7.1 Maximum Transmission Unit discovery
For a particular network path a maximum packet size will exist which can be sent along that path without being fragmented by the network. If this maximum transmission unit MTU is known, then a maximum ADU size can be chosen to fit within the MTU. Packet processing incurs overhead, so the more data that can be placed in one packet the better.

An ADU which is too large may be fragmented by the network causing additional overhead.

A mechanism is required for applications to discover the optimal MTU size such as proposed by Mogul and Deering (1990) for the network layer.

7.2 Checksum
In all the example applications the transport protocol has been UDP. UDP supports an optional checksum, however there is no mechanism for disabling the checksum on a per "connection" basis. The computing of the checksum may not in itself be a heavy computational process but reading all the data in a packet does cause a memory bottleneck.

It should be possible to disable checksumming on a per "connection" basis to allow the application to perform its own integrity checks or not.

An API should be provided to allow the checksum value to be passed across the interface allowing the application to roll in the checksum validation with its other operations.

8 CONTROL FUNCTIONS
A set of control functions should be provided for application use.

8.1 Retransmission Timeout
For any reliable protocol there is a requirement to occasionally retransmit lost data. It is important both that the data is retransmitted at an appropriate time and that during periods of congestion, an exponential backoff strategy is used. The calculation of an appropriate retransmission time and exponential backoff under conditions of congestion have been extensively covered in the literature (Braden, 1989).

In order to aid application builders and to reduce the possibility of congestion with a future generation of protocols a function should be provided to calculate the retransmission timeout. Such a function is used in the slogin program.

8.2 Rate control
Certain protocols use rate control to regulate the flow of packets onto the network. The MMD protocol uses rate control to keep the impact on the network of multicast mail delivery to a minimum. A control function could be provided that regulated the flow on packets onto the network.

8.3 Sequence space management
All the applications considered use sequence numbers to allow packets to be distinguished from each other. The sequence number is used for ordering and to detect loss. Most, if not all communications protocols use sequence numbers, a component which aided in the tracking of sequence numbers; to aid with ordering, reliability and reassembly would be extremely useful.

9 DATA FUNCTIONS
As well as some common control functions, some common data manipulation functions have been identified. The interface to the data manipulation functions should be standardised so that they can be used to build integrated loops.

9.1 Data integrity
If as in the example applications cited, we wish to roll the UDP checksum into the main processing loop then the checksum routine should be provided. If we wish to use our own checksum/crc mechanism then it should be possible to disable the standard checks and replace them with an application specific checksum/crc.
9.2 Compression
The vid application uses the h.261 standard, for its transmission of video it would not seem unreasonable that a module could be provided to do all the decompression into a X output buffer.

9.3 Encryption
The slogin program uses encryption as do other applications another set of common data transformation functions would be encryption functions.

10 CONCLUSIONS
A set of diverse applications were handcrafted in order to make best use of our knowledge of networks and the requirement of the applications. The aim was to identify common components which could be provided to a future generation of applications. The components may either be automatically combined or used by application builders as building blocks. As well as identifying common control and data manipulation functions some modifications to the communications API have also been suggested.

With this set of components and API additions it should be possible to better exploit ALF and ILP concepts.

As well as identifying common components of the diverse applications it has been shown that in all three cases it is possible to build a better optimised communications protocol when the application requirements are considered, than by using an off the shelf protocol such as TCP.

The lessons learned from implementing protocols such as TCP should not be lost. The calculation of retransmission timeouts and delayed ack policies are still useful in new protocols such as the slogin protocol.

Building a new protocol for each new application is time consuming and error prone, as more components of a communications system are identified they can be provided as library functions or as extensions to the communications API.

A common set of data manipulation functions have been identified however an open issue is the form of the interface to the data manipulation functions. If there is a requirement to achieve ILP and several data manipulation functions need to be combined then the data functions may need to be provided in a macro form as in (Abbot and Peterson, 1993) in order to achieve integration.

REFERENCES

GHOSH, A., HANDLEY, M., WANG, Z. and CROWCROFT, J. (1994): Integrated layer video decoding and application layer framed secure login-general lessons from two or three very different applications. In First International HIPPARCH workshop.

BIOGRAphICAL NOTES
Atanu Ghosh is a Research Fellow in the Department of Computer Science, University College London. He has a BSc(Eng) from Queen Mary College, University of London 1983 and a MSc in Data Communication Networks and Distributed Systems from University College London 1991. He has worked in industry on operating systems and networking. His current research interest is exploiting Application Level Framing concepts for the next generation of networked applications.

Jon Crowcroft is a professor of networked systems in the Department of Computer Science, University College London, where he is responsible for a number of European and US funded research projects in Multi-media Communications. He has been working in these areas for over 15 years. He graduated in Physics from Trinity College, Cambridge University in 1979, and gained his MSc in Computing in 1981, and PhD in 1993. He is a member of the ACM, the British Computer Society and the IEE and a senior member of the IEEE. He is general chair for the ACM SIG on COMM. He is also on the editorial teams for the Transactions on Networks and the Journal of Internetworking. With Mark Handley, he is the co-author of WWW:Beneath the Surf, and of the Open Distributed Systems, both published by UCL Press.
This paper describes the design and prototyping of a compiler tool for the automated implementation of distributed applications: ALFred. This compiler starts from the formal specification of an application written in ESTEREL, and then integrates end-to-end communication functions tailored to the application characteristics described in the specification; it finally produces a high performance implementation. The paper describes the communication architecture associated with our approach. The compiler is divided into two main parts: a control compiler also called ALF compiler; and a data manipulation compiler (the ILP compiler) that combines data manipulation functions efficiently in one loop. The ALFred compiler has been designed to allow the development and the analysis of non-layered high performance communication architectures based on ALF and ILP.
tion of data manipulation functions introduced by the ALF compiler for transmission control purpose (i.e. checksum, encoding), produces an efficient implementation organised in an ILP loop. A stub compiler then produces a base frame for the communication over the network.

Finally, a classic C compiler is used to link files produced by both the ALF and the ILP compilers to the original user C procedures.

We explain in section 2 our approach to efficient communication protocol architecture. We introduce the notion of "application QoS". Section 3 describes the ALFred compiler. Section 4 presents, both from a quantitative and a qualitative point of view, early performance evaluation of the ALFred protocol compiler. A detailed description of the ALFred compiler and its environment is provided in (Braun, 1996).

Figure 1: Structure of the ALFred Protocol Compiler.

2 COMMUNICATION SYSTEM ARCHITECTURE

2.1 QoS model

Today, two end-to-end transmission control protocols are generally used:

- Unreliable protocols (UDP). This type of protocol can be used when the data transmission does not require any reliability, or when the application provides itself end-to-end transmission control. This is the case of most of the multimedia applications that are available in the Internet (VIC, VAT, IVS, etc).

- Fully reliable protocols (TCP) where both reliability and order are provided. Application can consequently rely on the protocol. The application that needs reliability, and that has no constraints in delay uses this type of protocol (FTP, electronic mail, RPC, etc).

These two extreme protocols were sufficient when application requirements were limited. But complex multimedia applications, carrying voice, video and data, cannot be satisfied with these two protocols. They need protocols that are tailored to each data flow characteristics. TCP is too reliable for audio and video, and UDP not reliable enough for classic data. Most of the multimedia applications that are available on the Internet (VIC, VAT, IVS, etc) have their own own transmission control which is integrated to the application and use a UDP over IP based communication support (not reliable, not connected).

To obtain a fine-grain classification of end-to-end transmission control protocols, we have decomposed transmission control protocols in independent functionalities (e.g. acknowledgment, flow control, retransmission, etc). Functionalities and mechanisms are selected and associated to the application for each Application Data Unit (ADU) flow. Based on this three axes representation, we have defined three "application QoS" parameters. These parameters will be used to annotate the application formal specification in order to allow the ALF compiler to select and integrate dedicated and efficient communication facilities to each type of ADU flow within a distributed application:

- Ordering (or resequencing) corresponds to the order in which the data have to be delivered to the user. There are two classic notion of order: "in-sequence" and "out-of-sequence" delivering. We have defined two total ordering relations on ADU designed by the sending application.

  - Tx is the order of transmission. The transmission order is chosen by the sending application by association of a number to each ADU. In-sequence" ADU delivering corresponds to delivering in the transmission order.

  - Rx is the order of reception at the receiving entity. The reception order is the order in which ADUs are received from the network. Delivering ADUs in the receiving order means "out-of-sequence".

We have defined two ordering modules:

  - Tx.ORDER delivers the ADUs in the transmission order. This is the classic re-ordering function found in TCP or OSI TP4 for example.

  - Rx.ORDER which delivers the ADUs in the Rx order, or as they are received from network. This is the classic out of sequence delivering found in UDP.

- Reliability (or error control) is the mechanism used to detect and possibly correct lost ADUs. This parameter can take two values:

  - Not reliable that assumes no retransmission.
Fully reliable that means all ADUs will be delivered. When fully reliability is chosen, a retransmission module is integrated to the application to control the reliable delivering of the concerned flow of ADUs. Retransmission can be based on various type of mechanisms (positive, negative, selective acknowledgment) following the nature of the application and of the network.

Traffic characterises the real-time capabilities of an ADU flow. This parameter is concerned with the type of congestion control, access control, and flow control mechanisms that will be used to transmit an ADU flow. In our first ALFred prototype, there are two different values for this parameter: No real-time traffic and Real-time traffic. A real-time traffic will be controlled by a rate control mechanism, when non real time traffic will be controlled by a window flow control. But this is too weak for multimedia application and we intend to define a larger range of mechanism in the future, eventually modifying the values and the semantic of the Traffic parameter.

These three parameters allow an application designer to design eight different protocols.

2.2 Related works
Other research groups are currently working on the automated design and implementation of communication subsystems tailored to application requirements. We can mention ADAPTIVE (Schmidt, 1993), Da Capo (Plagemann, 1992), UTS (Richards, 1994), Partial Order Protocols (Diaz, 1995), and USC (O'Malley, 1994). The proposed solutions include developing general purpose protocols that allow flexibility. However these solutions are not operating system independent because the implementations are either part of the kernel, or a server within a micro-kernel based operating system. More details are given on two of these experiences:

- Da CaPo (Plagemann, 1992) is a more advanced tool for dynamic configuration of end-to-end transmission control protocols tailored to the application characteristics. A complex heuristic is used to design an independent control automaton for the end-to-end transmission control protocol (called CoRA). There is no integration, and the 3 layer architecture is respected. ALF is not retained as a design principle, which makes more complex the design of the communication support (itself implemented in the kernel space of the host computer). Da CaPo proves that tailoring protocol to the application characteristics is efficient in case of multimedia applications.

- (Diaz, 1995) also proposes a layered system where classic protocols are used to transmit application data, and where the application automaton is used to synchronise the data received (or transmitted) on the various protocols. The concept of "partial order connections" which is used to optimise the transmission, is very close to ALF. (Diaz, 1995) uses classic transport protocols; in the case of the JPEG player, one could TCP to transmit the control tables, and UDP for the MCU packets. The application automaton is described using Timed Petri Nets. Protocols used, as well as the application automaton, are implemented in the kernel space of the host computer.

3 THE ALFred COMPILER
The ALFred compilers is a prototyping tool designed to make the development of distributed multimedia application easier and faster. There are two objectives behind ALFred design: first to allow the prototyping of new applications; second to analyse new communication architectures and new mechanisms with a minimal development overhead.

3.1 The ALF Compiler
3.1.1 ALF based communication architecture
The Application Level Framing model (Clark, 1990) questions the well-established layered models. ALF says that the applications should be involved in their network communication and that more transmission functionalities and flexibility has to be left to the applications. The basic idea behind ALF is that the communication subsystem has to adapt to the application needs, as opposed to the application having to adapt to the underlying protocol as it is the case with the layered architecture. ALF considers that the application knows better than the transmission subsystem how to process data when there is some loss or/and when data arrives out-of-order.

Consequently, the subsystem must know the semantics of the application and the ALF model proposes that the communication system processes Applications Data Units (ADUs) which are meaningful for the application and not some packets only specific to the protocol. The ADUs are autonomous data which represent at the same time the unit of transmission, the unit of control and the unit of processing. In this scenario, ADUs can be processed out-of-order by the application at the receiver side. (Chrisment, 1994) shows that ALF can provide some improvements on performance due to the possibility of exploiting out-of-order processing.

3.1.2 Formal specification in ESTEREL
The application must be described in such a way that it is possible to analyse its structure automatically. Using a formal language is the perfect solution; we have chosen ESTEREL (Berry, 1992; Diot, 1994). There is no information relative to the end-to-end transmission control facilities at this level of specification. Just a description of the control automaton of the application. The application designer will just have to define the Application Data Unit (or ADU) that will be exchanged between the application entities. The compiler guarantees that these ADUs will be transmitted in a constituent way, i.e. not segmented by the transmission control functions.

The parameters defined section 2.1. that are used to characterise each type of ADU, are introduced in the ESTEREL specification as annotation to the ADU declaration. These annotations do no modify the syntax and the semantic of the language.
3.1.3 Automated integration
The ALF compiler makes use of a dedicated library of communication modules (also written in ESTEREL) to construct an integrated specification of a distributed application. The construction of the integrated application specification is realised in two steps:

— The first stage of the ALF compiler, called parser, analyses the structure of the application specification (to obtain synchronisation between ADUs) and introduce, for each type of ADU, a dedicated end-to-end control based on the QoS parameters found in the specification. Simultaneously to this parsing step, ADUs are encapsulated with transmission control information to be carried on the network. The resulting data units are called NDU for Network Data Units.
— The ESTEREL compiler is then used to merge the various automaton in a single and efficient automaton. A code optimiser is then runned to arrange the code following the most frequently used path (Castel, 1995).

The innovative aspect of our approach is that a set of end-to-end transmission control mechanisms is automatically selected for each type of ADU, and then integrated to the application control automaton. There is no more layered architecture between the application and IP. The result of this integration is shown Figure 2.

This approach is optimal on different aspects:
— it reduces the number of state to maintain,
— it removes interfaces between concurrent layers,
— only one “connection” (and the associated context) has to be managed.

![Classic architecture](image1)

![Integrated architecture](image2)

Figure 2: Transmission control modules integration.

This architecture has an immediate consequence: it allows the application to control resource using a single “session” identifier. The same identifier can be used by the underlying network.

3.2 The ILP Compiler
The ALF compiler generates the software implementing the automaton for an application-specific protocol. However, control processing is only one task of protocol processing. A second task, which has a particularly high impact on the performance of a protocol implementation, are the so-called data manipulation functions. These functions handle standard tasks in protocol software such as calculating a checksum, converting between different data representation formats or compressing and decompressing data.

In the ALFred system, the code for data manipulation functions is generated by a tool called the ILP compiler. This tool takes as input a specification of the structure of all messages exchanged by the application. When generating the code for data manipulation functions, the compiler automatically applies a code optimisation technique called Integrated Layer Processing.

To realise the functionality of an ILP compiler, we modified an existing stub compiler for generating marshalling code, the INRIA stub compiler (ISC) (Hoschka, 1994), so that it also produces ILP code, using the integration strategy described in (Braun, 1995). ISC extends other ILP approaches (e.g. (Abbott, 1993; Gunningberg, 1991)) by fully integrating data manipulation functions with marshalling code. A particular advantage of ISC when compared to other stub compilers is that it basically uses the C programming language as interface definition language (extended by a small set of annotations). The advantage is that the user does not have to learn a completely new language in order to use ISC.

3.3 ALF and ILP compilers in the ALFred Framework
The environment achieves the interaction between a synchronous behaviour represented by the integrated specification described in ESTEREL and the asynchronous world represented by the external environment (the user and/or the network).

This execution environment is composed of (see Figure 3):

— The control part implementing the integrated finite state machines. These automaton are directly generated by the ESTEREL compiler.
— The Network Data Unit (NDU) manipulation part implemented by the stub routines which are generated by the ILP compiler. These stub routines convert data into an appropriate network representation format (marshalling) and convert from the network data format to the local data format (unmarshalling). The network format used by the ILP compiler is presently XDR. The stub routines have been extended to integrate data manipulation processing. Several data manipulation like encryption, checksum and marshalling may be performed within a single loop.
— An asynchronous/synchronous interface (or Input Events Collector) collects events from the external environment (User/ Network) and generates signals to the automaton.
— A synchronous/asynchronous interface (or Output Events Collector) collects signals from the automaton and generates signals to the environment.

The interaction between the control automaton and the stub routines is achieved via the Input and Output Events Collectors (Figure 3). The incoming packets (or NDUs) are
received by the Input Events Collector which unmarshalls the NDU structures and applies the ILP loop. Then the Input Events Collector analyses the contents of the ILP result structure and generates the appropriate signal to the automaton. If something wrong happens during ILP loop, i.e. the data manipulation results differ between sender and receiver, the routine returns an error code. Some ILP errors have indeed to be signalled to the application (for example during decryption processing). The outgoing packets (NDU) are sent by the automaton through an emit statement to the Output Events Collector which marshals the NDU and applies the ILP loop.

![Diagram](image)

**Figure 3: Execution Environment.**

### 4 PROTOTYPE EVALUATION

To evaluate ALFRED, a client-server based JPEG (Wallace, 1990) image player has been implemented. The JPEG image is decomposed into a set of ADUs (Application Data Units) that can be received and processed out of order. However, the quantisation and Huffman tables must be transferred in order from the server to the client prior to the main image transfer. Image related ADU can be transmitted and delivered out of sequence. In other words, the level of reliability required for the tables is different from the rest of the image.

Two versions of this application have been implemented:

- A hand-coded version based on ALF (called ALF) that runs over its own protocol. This protocol (called TPALF) is a user-level TCP-like protocol that runs over UDP/IP.
- An automatically generated version using ALFRED. The ESTEREL version of the JPEG player runs directly over UDP/IP.

The experiments analyse the efficiency of the automated approach. The throughputs represent the amount of compressed image data divided by the time spent to transmit it. Experiments were performed on a lightly loaded live Ethernet (labelled Local Ethernet on next tables) and between France (INRIA) and Australia (UTS) (labelled Internet on next tables). Experiments were performed using Sun 10 workstations running SunOS (the average value of 100 measures is given).

Three issues concerning the automated JPEG player implementation can be analysed:

#### The design issue

The automated approach improves the flexibility and modularity of ALF based programs. This is best illustrated by an example. In the hand-coded implementation, the TPALF protocol is used during all the life of the application. The TPALF protocol allows out of order delivery of data. This is beneficial during the actual image transfer but goes contrary to design modularity, as the application must understand the protocol and explicitly reorder the JPEG table specifications. The protocol specified by ESTEREL changes during the life of the application, providing ordered delivery for the table specifications, and allowing out of order delivery of the image ADUs. Thus the application software was not required to implement any communication protocol functionality resulting in an improved software structure. The generalisation of this result is that if a protocol is not able to re-configure its functionality during the life of a connection (using ESTEREL or by any other means) then it should be designed to provide a level of functionality somewhere between the highest and lowest levels required during the life of a connection. If the protocol always provides the highest level of functionality, then the benefits of removing protocol functions cannot be exploited when the extra functionality is not required (for example forcing data to be ordered when the application no longer requires it). Always providing a lower level of functionality implies that the application is responsible for the additional protocol functions when required (for example the application must re-order some of the data).

Moreover, the states number of the integrated automaton is less than the sum of the states numbers of automata used for integration. For example, the client specification has five states and the selected communication modules represent 32 states when compiled separately. After integration, the ESTEREL description including application and communication modules is composed of 12 states.

#### The performance issue

We compared the C code generated by the ESTEREL compiler with the ALF hand-coded implementation. The protocol specified in ESTEREL did not implement the slow-start algorithm but used a simpler flow control; the acknowledgment packets are generated after each 4th ADU and the window size has been fixed to eight. In order to obtain a fair performance comparison, we modified the initial TPALF protocol so that both implementations use the same flow control parameters.

The results of this comparison are given in Table 1 and show that the choice of a formal and automated approach does not imply bad performances. Even, we observe that the ESTEREL automated implementation has a higher perform-
ance (20%) than the hand-coded implementation over an unreliable network like Internet. The better reactivity of the automated code, due to the optimised automaton produced by the ESTEREL compiler, improves the out-of-order processing and better benefits of the ALF concept. Over a more reliable network (like Ethernet), the ESTEREL automated implementation remains still better. The ESTEREL specification allows a better integration of the transmission control into the application. In the resulting automated code all protocol interfaces are suppressed except the user-kernel interface. Further experiments over large bandwidth networks (FDDI/ATM/Ethernet 100 base T) are foreseen to analyse the performances when the bottleneck is due to the protocol processing.

Table 1: Throughput between hand-coded and ESTEREL versions

<table>
<thead>
<tr>
<th></th>
<th>Ethernet</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALF</td>
<td>7.39 Mbits/s</td>
<td>51.3 Kbits/s</td>
</tr>
<tr>
<td>ESTEREL</td>
<td>7.42 Mbits/s</td>
<td>62.1 Kbits/s</td>
</tr>
</tbody>
</table>

The code issue
In the automated implementation, a dedicated user level transmission control protocol has been added to the application specification. The size of the server’s specification changed from 74 to 248 lines after parsing, while the client’s specification changed from 94 to 240 lines of ESTEREL code. On the automaton aspect, a two states automaton is produced for the server specification, and five states for the client specification. After protocol integration, both server and client are made of five states.

ESTEREL produces a code which is comparable in size to the code written in C language. The executable code of the receiver side is even smaller (4% on 200.000 bytes). This can be explained because, in the ALF hand-coded version, the TPALF protocol is implemented as a user-level library containing all functions even those never used either by the client or the server side. The automated approach allows to keep a certain level of modularity while producing a protocol more adapted and integrated to the application.

5 CONCLUSION
It has been demonstrated in this paper that the automated integration of transmission control functions in an application formal specification is possible in practice. Performance results confirm that, in term of code organisation, size, and efficiency, the ALF automated approach is almost as efficient as the hand-coded approach. Using this approach, a completely automated and efficient implementation of distributed applications is possible.

We are now working to a second prototype of the ALFred compiler that will be an enhanced version of the one designed in this document on many aspects:

- Protocol tailoring will be improved from experiment results and also from other research results. New functionalities will be added, as well as new control mechanisms (access control, congestion control, etc).
- ALFred will be extended to serve a larger range of applications, including multipoint multimedia applications.
- The notion of “application adaptivity” will be introduce to the application QoS. This will make possible the automated design of adaptive application using ALFred. Adaptiveness will be able to adapt to resources available on the network, even in association with guaranteed bandwidth services.
- Operating System dependencies are being analyse to determine what are the OS functions required, and how ALFred can be made independent of the host Operating System (or mostly independent).
- Performances of ALFred will be improved. The problem of performance will be analysed from the code generation aspect, the ALFred overall architecture, and the execution environment including interfaces and real-time kernel support.

The goal of ALFred is not to design a compiler tool, but to permit the prototyping of new applications and to analyse new communication architectures and new mechanisms with a minimal development overhead.

6 REFERENCES

BIOGRAPHICAL NOTES

Torsten Braun (M'94, ACM '96) received his diploma degree and his doctoral degree in computer science from the University of Karlsruhe, Germany, in 1990 and 1993, respectively. From 1990 to 1994 he was a research assistant at the Institute of Telematics, University of Karlsruhe. From 1994 to 1995 he has been a visiting scientist for one year at the Institut National de Recherche en Informatique et en Automatique (INRIA) in Sophia-Antipolis (France). Since September 1995, he is a guest scientist at the IBM European Networking Center in Heidelberg (Germany). His main research interests are the design and the implementation of high performance transport systems for multimedia communication.

Isabelle Chrisment received her MSc degree in computer science (communication networks) from the University of Nancy in 1987. Since February 1993, she is a PhD student at INRIA Sophia Antipolis, France, working on the study and the development of distributed applications within the Application Level Framing Architecture.

Christophe Diot (ACM '94) received his PhD in Computer Science from INP Grenoble in 1991. He is a research scientist at INRIA Sophia Antipolis, working on new architectures for communication subsystems (Design of communication support for time-constrained multimedia applications, Implementation of Adaptive applications). He is French representative in the COST 237 European project (Multimedia Telecommunication services). Diot is also the General Chairman of ACM SIGCOMM '97, to be held in Cannes in September 1997. His WWW server is http://www.inria.fr/rodeo/

Francois Gagnon received his MSc from INRS-Telecommunications in Montreal in 1995, where he studied formal verification of communication protocols. He is currently a PhD student at INRIA Sophia Antipolis working on the automatic generation of tailored protocols for multimedia applications.

Laurent Gautier received his MSc from University Pierre et Marie Curie, in Paris, in 1995. He has designed the first prototype of the ALP compiler. He is currently a PhD student working on communication mechanisms for distributed games in the RODEO project.
A Comparison of Automatic Protocol Generation Techniques

R De Silva
L Dairaine
A Seneviratne
M Fry
University of Technology, Sydney
School of Electrical Engineering,
PO Box 123, Broadway, NSW 2007 Australia

Due to the increasing complexity of applications and the availability of high speed networks, classical protocols have become the main bottleneck in communication systems. Although tailored protocols are able to respond to the needs of a given application, their development is expensive in terms of time and effort. An automatic protocol generation environment is most desirable. Two approaches currently used are the stub compilation and the runtime adaptive techniques. We have studied these two approaches and the behaviour of the resulting tailored transport protocols. Relative performance, comparisons and discussions about these two approaches are presented in this paper.

Keywords: Tailored Protocol, Automated Implementation.

1 INTRODUCTION
The development of flexible and efficient communication protocols is an important step towards the realisation of high performance distributed applications running on new high speed networks. Developments in high speed networking are influenced by two major factors: the increasing capability of end-systems and communication networks, and the diversity and dynamism of new applications. Recent networks such as FDDI and ATM allow for high speed transmission of digital data and have low latency characteristics, thus enabling the design of new types of applications such as multimedia. High level protocols, like TCP and TP4, have not evolved to take developments in applications and networking into consideration, resulting in them becoming a bottleneck in the communication system (Clark and Tennenhouse, 1990).

Protocols can be tailored to application needs and network conditions. Different approaches can be taken to tailoring protocols. One method is to hand-craft the tailored protocol. This approach often leads to high performance protocols for a given application and network, but demands a lot of effort and time to design, implement and maintain. The alternative to hand-crafting protocols is to automatically generate application specific protocols. A number of research projects are currently in progress, studying the possibility of automatically implementing a protocol based on a specification of application requirements and network resources. This paper reports a comparison of two approaches used for the automatic generation of tailored protocols.

The paper is organised as follows. In section 2 we study the evolution of protocol development, discussing the main techniques used in protocol design. Two automatic protocol tailoring approaches — namely compilation-based and runtime adaptive — will be studied in section 3. Section 4 presents our testing environment and results, both in terms of quantitative performance measurements and methodological aspects. Concluding remarks are given in section 5.

2 TAILORING PROTOCOLS IN HIGH SPEED ENVIRONMENTS
Protocols should be designed to handle the varying requirements of multimedia applications and to take into consideration the underlying network support. The main techniques that can be used to tailor a protocol are: the optimisation of a current implementation, the development of new mechanisms, and the development of new protocols.

Implementation optimisations can be applied to existing protocols to improve performance. Such techniques do not change the functionality of the protocol but simply reduce the processing cost of the protocol.

An alternative technique is to develop new mechanisms. This is often achieved by re-engineering an existing protocol to reduce a bottleneck that arises out of a particular characteristic of the operating environment. These implementations often present a large number of options. Only a limited set of these options are used by a particular application.
The third technique is the development of new protocols built to exactly match a given application's requirements. This technique would normally also utilise the previous two techniques discussed above. The main drawback of this method is the development cost for a specialised protocol. Automatic protocol generation permits a high level of tailoring without the high costs of time and manpower.

3 AUTOMATED COMMUNICATION PROTOCOL GENERATION

Automated tailored protocol generation can be achieved by defining, for a given application, the functionality the protocol should provide and the associated mechanisms. The overall process involves three basic tasks: specification, selection and synthesis.

During the specification phase, the application developer lists all information that characterises the application and the environment. This list should contain all the relevant information needed by the Automated Communication Protocol Generator (ACPG) to create the appropriate tailored protocol. Such a specification should contain, for example, the structure and characteristics of the data being transferred, ordering constraints, reliability, timing criteria, possibilities of having self-contained data packets and possible integrable processes.

The selection of mechanisms is the second phase of the automated process. Using information that characterises the application, the ACPG decides the overall functionality required to build the tailored protocol. If the protocol is intended to be dynamic, then decisions on when to switch protocol functionality can be decided at this stage.

Finally, the synthesis phase involves the implementation of the protocol. It has been shown that, for efficient implementations, the principles of Application Level Framing (ALF) and Integrated Layer Processing (ILP) should be adopted (Clark and Tennenhouse, 1990). The implementation can be static or dynamic. Dynamism can be introduced by dynamically linking in and out the protocol functions as required, or statically implemented in a state machine which changes states when changes in protocol functionality are required. The latter can result in “code bloat” if a large number of dynamic states are defined.

Currently there are a number of projects being conducted on automated approaches. These include DaCapo (Vogt, Plattner, Plagemann and Walter, 1993), F-CSS (Zitterbart, Stiller and Tantawy, 1993), ADAPTIVE (Schmidt, Box and Suda, 1992), the Runtime Adaptive Approach (Universal Transport Service) (Richards, 1995), and the Stub Compiler (STRL) (Castelluccia and Dabbous, 1994). The first four models provide runtime configuration while the fifth model provides configuration at compilation time. The first three approaches tailor the whole communication environment while the remaining two create application tailored protocols. The rest of this paper is devoted to these two last techniques, which have been selected to contrast the runtime and compilation approaches.

3.1 Runtime Adaptive Approach

The runtime adaptive approach has been developed at UTS (University of Technology, Sydney) in Australia. As shown in Figure 1, the conceptual architecture of the runtime adaptive model follows the general model for ACPG.

![Figure 1: The runtime adaptive model.](image-url)

The realisation follows the 3 phases as defined above. At runtime, the application indicates its requirements to the Functionality Selector, for example via a QoS management entity. The Functionality Selector then determines the protocol functions that will be required to satisfy the application’s requirements. A profile is then generated that indicates to the Synthesis Engine the optimal choice. The Synthesis Engine then takes into account the environment status (e.g., network status and system load) and chooses the appropriate mechanisms to provide the requested functionality. It then uses implementation techniques to optimise the transport system’s performance. In addition, the Synthesis Engine continuously monitors the status of the network and host system and, where possible, dynamically chooses the protocol mechanisms that will best suit the given conditions.

The runtime adaptive model differs most from the stub compiler model in its inherent dynamicity. The configuration engine creates/modify the protocol as necessary following changes in any of its inputs (network status, profile, etc.). Dynamicity is achieved by dynamically linking in and out the appropriate protocol functionality as determined by the Synthesis Engine.

The adaptive approach takes into account the ALF and ILP principles, although our implementation currently does not apply ILP.

3.2 Stub Compilation Approach

The stub compilation approach has been developed at INRIA (Institut National de la Recherche en Informatique et
A COMPARISON OF AUTOMATIC PROTOCOL GENERATION TECHNIQUES

Automatique) in France. This approach is a preliminary step in the development of a new generation of remote procedure call models (Diot, Chrisment and Richards, 1995). In this model, a distributed application can specify its own communication requirements, which are to be associated to a dedicated communication system. This is realised by means of an application specification, which is used by a protocol compiler to integrate communication facilities with the application by generating client and server stubs.

The specification step is achieved using both Esterel and C code. Esterel (Berry, 1992) specifies the control and synchronisation aspects of the communication, while data structures and application software are directly coded in C. More specifically, the Esterel specification describes the application's behaviour, and the level of reliability required for the transmission of the Application Data Unit (ADU) — the smallest unit of data that the application can process to completion. The services required are described by a set of predefined Esterel signals expressed in the specification, such as selective_retransmission, flow_control, checksum, etc.

The selection step is realised through the parsing of the specification as shown in Figure 2. From the Esterel description of the application's behaviour, the parser extracts possible synchronisation points and any parallelism that exists between the different modules that compose the description. The parser extracts possible synchronisation points and any parallelism that exists between the different modules that compose the description. These synchronisation points will be used in the synthesis stage to construct an efficient implementation. The predefined Esterel signals present in the application specification are extended by the parser to integrate the requested communication facilities. The result of this step is the integrated specification, still expressed in Esterel.

The new Esterel specification is then compiled through an Esterel compiler which produces C-code for a finite state machine defining the different protocol states. This C-code is then combined with the appropriate application code and the appropriate protocol functions to create the executable application. The C and Esterel compilers form the implementation stage of the ACPG model.

The main strength of the stub compiler model is that it provides a high degree of "tailorability". The main reason for this is the use of Esterel — a formal language, which allows for an accurate description of the application's requirements. The stub compiler model can realise Application Level Framing. One weakness of this model is that the stub compiler is based on a state machine which changes state to provide different services.

Currently, this model does not properly consider dynamic adaption as it fails to take into consideration external factors such as network and system loads.

4 EXPERIMENTATION

To compare the runtime adaptive and stub compiler approaches, a common application was selected and two tailored protocols were generated. Quantitative and qualitative measurements of the resulting protocols' performance, and their development methodologies, were then evaluated in different environments.

4.1 The JPEG Image Server

The JPEG Image Server (JIS) is a client/server-based application allowing the visualisation of images stored at the server. JIS constitutes a system that is sufficiently simple to be ported quickly to several distinct platforms, and easily modifiable to run over different implementation environments, but is still sufficiently complex that it can benefit from application specific tailoring.

The main property of JIS that can be exploited by tailoring is its partial ordering of the JPEG image elements. The client does not require the ordered reception of image data messages. The only real ordering requirement is to receive the JPEG specification tables before the image data (these tables specify parameters of the compression such as quantisation). Then, during the data transfer phase, the communication protocol can present the image data blocks to the application regardless of the order in which they are received (this assumes that the application has structured its transmission according to ALF principles). Thus, JIS can be realised using two different protocols, namely one for the specification phase and another for the data transfer phase.

The runtime adaptive and stub compilation ACPGs were used to develop protocols tailored to JIS. The following sections discuss the results obtained from the comparative study of the two automated approaches. The protocols developed by the runtime adaptive and stub compiler ACPGs will be referenced as UTS and STRL respectively. The performance of the protocols is presented in terms of qualitative issues.

Figure 2: The stub compiler model.

The selection step is achieved using both Esterel and C code. Esterel (Berry, 1992) specifies the control and synchronisation aspects of the communication, while data structures and application software are directly coded in C. More specifically, the Esterel specification describes the application's behaviour, and the level of reliability required for the transmission of the Application Data Unit (ADU) — the smallest unit of data that the application can process to completion. The services required are described by a set of predefined Esterel signals expressed in the specification, such as selective_retransmission, flow_control, checksum, etc.

The selection step is realised through the parsing of the specification as shown in Figure 2. From the Esterel description of the application's behaviour, the parser extracts possible synchronisation points and any parallelism that exists between the different modules that compose the description. These synchronisation points will be used in the synthesis stage to construct an efficient implementation. The predefined Esterel signals present in the application specification are extended by the parser to integrate the requested communication facilities. The result of this step is the integrated specification, still expressed in Esterel.

The new Esterel specification is then compiled through an Esterel compiler which produces C-code for a finite state machine defining the different protocol states. This C-code is then combined with the appropriate application code and the appropriate protocol functions to create the executable application. The C and Esterel compilers form the implementation stage of the ACPG model.

The main strength of the stub compiler model is that it provides a high degree of "tailorability". The main reason for this is the use of Esterel — a formal language, which allows for an accurate description of the application's requirements. The stub compiler model can realise Application Level Framing. One weakness of this model is that the stub compiler is based on a state machine which changes state to provide different services.

Currently, this model does not properly consider dynamic adaption as it fails to take into consideration external factors such as network and system loads.

4 EXPERIMENTATION

To compare the runtime adaptive and stub compiler approaches, a common application was selected and two tailored protocols were generated. Quantitative and qualitative measurements of the resulting protocols' performance, and their development methodologies, were then evaluated in different environments.

4.1 The JPEG Image Server

The JPEG Image Server (JIS) is a client/server-based application allowing the visualisation of images stored at the server. JIS constitutes a system that is sufficiently simple to be ported quickly to several distinct platforms, and easily modifiable to run over different implementation environments, but is still sufficiently complex that it can benefit from application specific tailoring.

The main property of JIS that can be exploited by tailoring is its partial ordering of the JPEG image elements. The client does not require the ordered reception of image data messages. The only real ordering requirement is to receive the JPEG specification tables before the image data (these tables specify parameters of the compression such as quantisation). Then, during the data transfer phase, the communication protocol can present the image data blocks to the application regardless of the order in which they are received (this assumes that the application has structured its transmission according to ALF principles). Thus, JIS can be realised using two different protocols, namely one for the specification phase and another for the data transfer phase.

The runtime adaptive and stub compilation ACPGs were used to develop protocols tailored to JIS. The following sections discuss the results obtained from the comparative study of the two automated approaches. The protocols developed by the runtime adaptive and stub compiler ACPGs will be referenced as UTS and STRL respectively. The performance of the protocols is presented in terms of qualitative issues.
and quantitative measurements. Only prototype versions of both ACPGs are currently available. These were used to obtain the experimental results.

4.2 Qualitative ACPG Issues
The two major aims of an ACPG are to offer easy and rapid development of protocols and to produce high performance implementations. These two issues are discussed in this section.

Protocol Development Issues
From the user point of view, the runtime adaptive and stub compilation techniques take completely different approaches to the development of tailored protocols and provide different interfaces. The former is based on a protocol configuration using its own interface, which is currently based on a mix between a BSD socket interface and function-call interface. The latter provides a formal protocol description using both the Esterel and C languages.

The two ACPGs also differ in the manner of expressing application needs and in the management of protocol adaptations. With the compiler-based ACPG, the JIS application defines a protocol using an Esterel specification indicating the two different stages in the data transfer and their corresponding functionality. The parser and compiler stages of the ACPG then creates a finite state machine which will change states depending on input signals from the application. Changes of state occur at set points in the data transfer and are agreed at compilation time. We therefore do not require a meta-protocol to signal changes at runtime between the client and server.

Considering the JIS client side, the Esterel code is based on two main parts specifying the two phases. The first part basically waits for two specification tables, using a parallel construct. When both tables have reached the client, a signal causes the state machine to terminate the specification phase and begin the data transfer phase. This enables out-of-order reception of the different image data messages. From the user point of view, the image data ADUs are displayed as soon as they reach the client, resulting in a possible mosaic-style effect as the image is being constructed.

The runtime adaptive model protocol requires the application to request functionality through an interface. With the current prototype of the model, the application specifies the required functionality through the use of function calls. Since functionality changes occur at runtime, and because the protocol has no prior knowledge of any changes, we require a meta-protocol to co-ordinate the changes between the client and server. UTS currently realises the meta-protocol within the packet header. As a result, there will normally be a round-trip delay before functionality is switched.

To improve performance, instead of defining two stages of unordered transfer separated by a synchronisation point, the UTS protocol was defined as an ordered protocol for the first stage of the transfer (i.e. the specification stage) followed by an unordered stage for the image data. The change from ordered delivery to unordered delivery is a reduction in service. Hence it is unnecessary to stop the transmission of data until the functionality has changed. In contrast, if we were to change from unordered to ordered delivery, we would have to guarantee that every packet the user specified as ordered arrived in the correct sequence. When changing in the reverse direction we provide a higher level of service while the change in functionality occurs.

The user also specifies through a function call that they wish to use ALF. This ensures that the data written by the application through the socket interface will be handled as an integral unit by the communication system. This is required to enable unordered delivery of ADUs.

In both approaches the effort required by the user is minimised to defining the requirements of their application.

Resulting Code
Experiments were carried out on DEC 5000/240 Workstations using a dedicated Ethernet network. Experiments were executed over two different operating systems, namely Ultrix v4.3 and the Mach 3.0 micro-kernel.

With the Mach micro-kernel, TCP can be implemented using two possible methods. The first is the Mach 3.0 UX server, which provides a kernel level implementation of TCP/IP within the Unix Server. The second is a user level library implementation of TCP (Maeda and Bershad, 1992).

UTS is currently based only on the Mach 3.0 user level library implementation. Therefore the results for UTS were only measured for this implementation. The results for TCP (which was used as a benchmark) and STRL were obtained using all three platforms: Ultrix 4.3, Mach 3.0 using the UX server and Mach 3.0 using the user level socket library.

The code sizes of the resulting protocols varied. Table 1 shows the sizes of TCP, STRL and UTS in the different environments.

TCP code is optimal in all situations. The STRL code is significantly larger. This is because it includes its own protocol that runs over UDP/IP. This is in contrast to UTS, which replaces TCP. Hence for user level protocol implementations, UTS is the same size as TCP, while STRL runs above the user level implementation of UDP/IP adding another level of protocol functionality. The current version of UTS simply uses internal switches to change functionality. The final version will dynamically link the required functionality and hence the overall code size is expected to be smaller.

Table 1: Architecture-oriented sizes in kilobytes for the client and server executables.

<table>
<thead>
<tr>
<th>Client Sizes</th>
<th>TCP</th>
<th>STRL</th>
<th>UTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0 SC</td>
<td>539.6</td>
<td>819.4</td>
<td>539.6</td>
</tr>
<tr>
<td>Mach 3.0 UX</td>
<td>285.3</td>
<td>546.2</td>
<td>n/a</td>
</tr>
<tr>
<td>Ultrix 4.3</td>
<td>278.4</td>
<td>547.3</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Server Sizes</th>
<th>TCP</th>
<th>STRL</th>
<th>UTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mach 3.0 SC</td>
<td>510.6</td>
<td>802.6</td>
<td>510.6</td>
</tr>
<tr>
<td>Mach 3.0 UX</td>
<td>256.2</td>
<td>445.8</td>
<td>n/a</td>
</tr>
<tr>
<td>Ultrix 4.3</td>
<td>249.4</td>
<td>446.0</td>
<td>n/a</td>
</tr>
</tbody>
</table>
4.3 Quantitative Performance

Three experiments were carried out to capture how well the implementations of the JIS tailored protocols performed. The first compared the behaviour of the three protocols with various ADU sizes. The second compared STRL and TCP performance across the different platforms. The third showed the behaviour of the protocols when transmission errors are introduced.

Comparison of the Three Protocols

The Experiments were carried out on the Mach 3.0 Kernel using the user level implementation of TCP/UDP/IP for all the protocols. We experimented using two MCU ADU sizes (MCU = Minimum Coded Unit, an atomic unit of compression/decompression). One was the maximum size for a MCU ADU such that any ADU can be contained without IP fragmentation (i.e. 640 bytes). The other was the maximum size for a MCU ADU without it begin fragmented at the MAC layer (i.e. 1460 bytes). Table 2 shows the results of the first experiment.

The throughput results attained using the smaller MCU ADU size are low. This is because of inefficient usage of the underlying network. STRL is based on UDP, and hence a large number of UDP packets are sent across the network. Similarly, with TCP and UTS (which is based on TCP) the small sized MCU ADU are buffered to try and fill a packet, hence introducing delays. When we increase the MCU ADU size to the maximum for the network, we notice that there is a large increase in the throughput.

Cross Platform Comparisons

The second experiment compared the performance of the STRL protocol against TCP across the different platforms. The maximum size of MCU ADU was used for all experiments to achieve the best results. Table 3 illustrates the comparative behaviour of TCP and STRL across the different environments. With the exception of the Mach 3.0 UX platform, where there is a drop in performance for the STRL protocol, the two protocols have very similar performance.

With Mach for both socket code (SC) and UX server (UX), the results are at least 2Mb under those of Ultrix. This can be explained by the user level location of the transport protocols and the better implementation of the Ethernet device driver in UX (Witana, 1994).

Performance under Erroneous Conditions

The selective retransmission and ALF architecture of the STRL protocol leads to a very good behaviour in presence of transmission errors introduced by the network (both packet loss and bit error). Missing packet are requested when the receiver finds an out of sequence pattern, but this does not block the sending application. Thus the overall throughput can be estimated assuming that the transmission delay is just the time to send ADUs and retransmitted ADUs.

The same experiment with TCP gives poor results. TCP interprets packet loss to be caused by congestion, and therefore reduces transmission rate to allow the network to recover. TCP’s slow start mechanism is responsible for this behaviour.

5 CONCLUDING REMARKS

In this paper we have discussed the benefits of tailoring protocols to application needs. Currently, tailoring techniques are costly in terms of time and effort, and require highly skilled personnel. Automated protocol generation aims to keep the benefits of tailoring without the associated costs of development.

A number of different automated approaches for Automated Communication Protocol Generation have been proposed, of which we focused on two, namely runtime adaptive and stub compilation. These two ACPGs were used to build protocols tailored to a JPEG Server Application. Both approaches were able to create protocols with tailored functionality, as would be done by a hand-crafted implementation.

Our results show that the two tailored protocols have similar performance although the implementations differ. Both protocols provide similar functionality with the exception of error recovery mechanisms. The main difference that currently exists between the two approaches is the method by which functionality is requested and supported.

Comparisons between the automatically generated protocols and a hand-coded TCP implementation resulted in the same range of throughput over a number of platforms. This indicates that the automated approach does not interfere with the quality of implementation of the resulted protocols. In addition, some engineering concepts such as ILP could be taken systematically into account, yielding better implementations. In conclusion, it has been shown that automated protocols are a viable solution for the future.

6 ACKNOWLEDGMENTS

The authors would like to thank Isabelle Chrisment from INRIA (France) for her assistance in the JIS application. Laurent Dairaine thanks INRIA for its financial support.
REFERENCES


BIOGRAPHICAL NOTES

Ranil De Silva holds a BCompSci (Hons) from Bond University, Gold Coast. He began his PhD at the University of Technology, Sydney in August 1994. His primary research interests are high speed protocols and mobile networking.

Laurent Dairaine is currently Associate Professor of Computer Science at the Ecole Nationale Superieure d'Ingenieur de Constructions Aeronautiques (ENSICA) in Toulouse. He received a PhD in computer science from Pierre et Marie Curie University of Paris at MAS Laboratory in 1994. In 1994 and 1995 he spent a year as a visiting researcher at University of Technology, Sydney. His research interests include real-time and multimedia systems, protocol development and high speed networks.

Aruna Seneviratne has a BSc (Hons) from Middlesex Polytechnic in Electronic Engineering and a PhD from the University of Bath. He has worked in industry and academia. In 1993/4 he was employed by the MASI Laboratory in Paris for six months to work on the development of an end-to-end quality of service management scheme for distributed multimedia applications. He is currently an Associate Professor and is the Director of the Telecommunications group at University of Technology, Sydney. He has previously been awarded fellowships by Telecom Australia, British Telecom and INRIA.

Michael Fry received his PhD from the University of Sydney. He worked for a number of years in the computer industry designing and constructing real-time monitoring systems. He is currently an Associate Professor with the School of Computing Sciences at the University of Technology, Sydney. His active research interests include distributed multimedia systems, high performance protocol architectures, and nomadic computing.
Some Issues for Dynamic Synthesis of ALF/ILP Systems

Michael Fry
Atanu Ghosh
School of Computing Sciences,
University of Technology, Sydney
PO Box 123, Broadway, NSW 2007 Australia

ALF and ILP are powerful principles that guide the construction of application specific, high performance distributed systems. This paper explores some ideas for modular construction of ALF/ILP systems. We consider how flexible "filter pipelines" may be dynamically synthesised at run time, while achieving high performance. We further consider the application of ILP to a complex, distributed multimedia application. Our results illustrate some of the trade-offs in constructing adaptable, high performance systems.

1 INTRODUCTION
Consideration of the general problem of optimal protocol design and implementation has lead to the development of two significant principles: Application Level Framing (ALF) and Integrated Layer Processing (ILP) (Clark and Tennenhouse, 1990).

ALF and ILP are concise and powerful principles. The paramount aim of ALF/ILP is flexible decomposition. This dictates that engineering decisions should be left, as far as possible, to the implementor. She can then construct a communication profile that best suits the environment, i.e. the application behaviour and the network. The architectural principle of ALF recognises that the application is often the bottleneck in communication. Thus the application should be permitted to process data as soon as it arrives. However this only achieves efficiencies if the whole Application Data Unit (ADU) is available to the application from the network. Thus support for ALF should allow the application to discover and express the optimal ADU as a parameter to all protocol functions.

The engineering principle of ILP addresses the high cost of data manipulation. Ideally, an Integrated Layer Processing implementation will perform only a single loop over the data. Within that loop all data copying and all protocol functions from every layer that need to "touch the byte" will be performed. To achieve ILP we should provide options to integrate processing loops, and should minimise ordering constraints.

While it seems clear that ALF/ILP based communication systems can be built by hand for certain distributed applications, we would like to prevent system builders from repeatedly solving the same problems. This paper explores some ideas for modular construction of ALF/ILP systems. It is particularly focused on ILP, attempting to apply ILP to two challenging scenarios. Firstly, we consider the construction of flexible filter pipelines that are not only built from modular components, but are also dynamically adaptable. Secondly, to date ALF/ILP has only been applied experimentally to the construction of fairly simple communication systems. We explore the usefulness of ILP for a complex, multimedia application, and discuss some of the problems of modular solutions for these applications.

In the next section we present some broad ideas as to how modular, dynamic ALF/ILP systems may be constructed. The following section presents some initial results of experimentation into dynamic filter pipeline mechanisms. We then present a study of a complex multimedia application and discuss the possible application of modular ILP to this application. We then offer our conclusions.

2 SYNTHESISING PROTOCOL FUNCTIONS USING ALF/ILP: SOME IDEAS
Our broad aims are:
— to permit the (relatively) easy construction of communication systems by programmers, based on the principles of ALF/ILP, and which realise the performance benefits of ALF/ILP
— to achieve this construction with re-usable modules.
2.1 Smart Buffers

There is a danger that any set of tools we provide may restrict the application developer in certain cases. To guard against this we make the observation that communications is based around input and output buffers. So we build a smart buffer abstraction around input and output buffers. This is illustrated in Figure 1.

A smart buffer is used to fulfil three main functions. The first is to provide a framework around which ALF style applications can be built. The second is to aid with no-copy buffer management (Druschel and Peterson, 1993). The third is to provide a systematic way of using ILP.

The communications applications in which we are interested will all be either sending or receiving data; most will be doing both. So we suggest a smart buffer, which is an object oriented interface to the receive and transmit buffers. In a traditional communications infrastructure an application would provide a buffer into which data received from the network should be placed. In the case of smart buffers, a smart buffer is requested from the system and this handle is passed to a receive call, rather than an actual buffer address. A smart buffer has a set of data manipulation operations which it supports, most importantly it allows filters to be attached to the buffer. The filters can either be word filters (Abbott and Peterson, 1993) or, if appropriate, a filter which traverses the whole buffer. The filters can be either specified by the user or added by the system. For example checksum routines may be attached by the system. Another data manipulation function provided may be a copy function. When the copy function is invoked the various filters are invoked and the data referenced by the smart buffer is copied to the output location.

2.2 Control Processing

As well as aiding the processing of the data in buffers the smart buffer can also be used to realise the control processing which typically takes place in a protocol implementation. Packets in a network protocol have headers. Control filters can be used to add headers on transmission and to extract header fields on packet reception. Once the header fields have been extracted they can be converted to a canonical form which can be processed by the processing elements described in the next section.

2.3 Abstraction and Interoperability

The smart buffer abstraction gives a structure for incoming and outgoing messages. Together with this structure, and with the aid of header processing filters, it is possible to provide general processing elements which can operate on the smart buffers. An example of such a processing element would be a sequencer, which is responsible for the sequence space management of a protocol. For an ALF based video application missing packets can be tolerated, but the ordering is important. For an ALF based file transfer application ordering is not important but all blocks of a file must be delivered. For an ALF based remote login session both ordering and reliability are required. So a processing unit which provides sequencing can be variously configured for different applications.

With the aid of the smart buffer abstraction and some processing elements such as the sequencer it should be possible to build extremely flexible and optimised applications. As the filter interface to the smart buffer is well defined a library of operations to perform operations such as compression and encryption could also be provided.

3 MECHANISMS FOR MODULAR SYNTHESIS

We now consider some concrete mechanisms for the realisation of dynamic, modular filter pipelines. Abbott and Peterson (1993) have proposed word filters as a general purpose, modular ILP implementation technique. Word filters execute each stage of a pipeline for each 32 bit machine word of the input stream. However, different stages may operate on different data widths. In (Abbott and Peterson, 1993) the example of DES is given, which manipulates 64 bit quanta (in block mode). A DES word filter will maintain state such that it will only perform its manipulation on pairs of words. When the first word is input from the pipeline it is simply saved. DES
manipulation occurs when the second word is input, with the two word results being output sequentially to the next filter, e.g. a checksum word filter. This encourages all operands and partial results to remain in registers and/or cache. Abbot and Peterson demonstrate the use of word filters in an example pipeline. Performance is shown to be superior to serial implementation, even for worst-case cache behaviour.

The word filter idea is also used by Braun and Diot (1995). As with Abbott and Peterson (1993), word filters are implemented as source macros. This essentially means that protocol synthesis or binding occurs at compile time. One problem with the macro approach is that the code tends to lose modularity, becoming rather ugly and difficult to maintain. Dynamic, run-time synthesis would permit adaptation of the filter pipeline on the fly. This would be useful for handling network fragmentation of ADUs, or for breaks in the processing pipeline, for example delayed playout of audio or video.

Thus we have experimented with building a dynamically configurable filter pipeline out of modular components. For the time being we have ignored issues of data/control dependencies, such as have been identified by Braun and Diot (1995). Our focus is on the mechanisms for dynamic synthesis and their costs.

3.1 Dynamic Filters
The filters we have initially chosen are a simple byte swap (BSWAP), the SAFER K-64 encryption algorithm (Massey, 1994), and the TCP checksum (CKSUM). In our implementation the order in which the filters are called can be determined dynamically. As with word filters (Abbot and Peterson, 1993), a single variable DataWord is used to pass the current word between filters. The dynamic nature of the pipeline is achieved by putting each filter as one case in a switch statement. The control is achieved by having a vector of switch labels which are traversed sequentially. The original intent was to try and perform a computed goto within one function in order to jump between filters. Unfortunately C does not allow this construct so the next best solution was to use a switch.

In order for the filters to be totally dynamic, as well as the transform cases such as the BSWAP and the CKSUM, there is an initialise stage START which loads a word from the input buffer into the DataWord. There is also a final stage END, where the final output DataWord is stored into the output buffer.

3.2 Experimentation
We have measured the performance of our implementations on a DEC Alpha/AXP 3000 model 500. Our principal measurement tool is pixie — a profiling tool that provides the number of machine cycles executed by each basic block of code, down to the source statement level. Initial measurements are in Table 1.

Our experiment simply processes a 4096 byte buffer from an input buffer to an output buffer. We compare the following cases. The Copy case is a control for our experiment. It simply copies 4096 bytes from the input buffer to the output buffer.

In the Traditional case we combine the three transforms in the traditional way, i.e. totally processing each buffer, before performing the next transform. In the Integrated case we integrate the transforms using the word filter macro approach of Abbot and Peterson (1993), i.e. a static integration of the transforms. Filter is as described above: a single function which contains all the code for the transforms. However the order of the transforms is determined dynamically.

Table 1: Filter Comparison.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Cycles</th>
<th>Calls</th>
<th>Loads</th>
<th>Stores</th>
<th>Loads + Stores</th>
<th>Cycles normalised</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>9158</td>
<td>35</td>
<td>1415</td>
<td>1197</td>
<td>2612</td>
<td>1.0</td>
</tr>
<tr>
<td>Traditional</td>
<td>177336</td>
<td>555</td>
<td>33983</td>
<td>14070</td>
<td>48053</td>
<td>19.36</td>
</tr>
<tr>
<td>Integrated</td>
<td>85629</td>
<td>39</td>
<td>12470</td>
<td>6898</td>
<td>19368</td>
<td>9.35</td>
</tr>
<tr>
<td>Filter</td>
<td>160404</td>
<td>39</td>
<td>31425</td>
<td>8951</td>
<td>40376</td>
<td>17.51</td>
</tr>
</tbody>
</table>

3.3 Discussion
We observe that the (statically) Integrated implementation executes approximately half the number of cycles of the Traditional one. We infer therefore that it will execute about twice as fast. This has been verified by measuring the real time of execution. Our measurements of real time show performance scaling with the number of instructions and the number of loads and stores, as seen in Table 1. This tends to support Braun and Diot (1995), who conclude that the benefit of ILP is primarily due to reduced memory access, rather than enhanced cache behaviour.

Nonetheless it is clear that our dynamic pipeline is not delivering the performance gain of the statically integrated case. Table 2 shows an analysis of cycle consumption by the various components of Filter. The first five lines are the filters — the three transforms plus the start and end filters. The following line represents the processing required to accommodate different datum widths for the various filters. For example, SAFER requires an 8-byte datum, whereas the checksum requires only 2 bytes. This necessitates additional state and control logic as, for example, the SAFER filter accumulates its input.

The final line is the control overhead of the switch — most of this is in fact consumed by a single C “case” statement that switches between the filters. It can be seen that the filters themselves account for 51 percent of the cycles. It is surely no coincidence that the statically integrated pipeline consumes approximately half the cycles of the dynamic pipeline. Put another way: the additional control overhead of the dynamic pipeline seems to account almost exactly for the observed performance hit.

Table 2: Filter Cycle Breakdown.

<table>
<thead>
<tr>
<th>Component</th>
<th>Cycles</th>
<th>Calls</th>
<th>Loads</th>
<th>Stores</th>
<th>Loads + Stores</th>
<th>Cycles normalised</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSWAP</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAFER SK-64</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CKSUM</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datum Width</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Overhead</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A key question then is whether this control overhead can be reduced. We observe that the critical filter switch statement is in fact being executed many more times than the actual number of filters. This is caused by the varying data width problem — the pipeline is partially executed to the SAFER filter, which requires more than a word, and is then executed again to supply the second word.

This problem could be overcome if the datum width were the same for each filter, and equal to the maximum width required by any filter in the pipeline. This would also help reduce the “Datum Width” cycles in Table 2. However generality would be compromised. The “word filter” idea maintains a common, uniform interface between filters. This enhances modularity. Furthermore it may not be possible to build lightweight filters that can adapt to varying datum widths. This requires further research.

Likewise the C case statement used to multiplex the filters is very expensive. Further experimentation is required to try to find lightweight control mechanisms. Thus the aim of modular, dynamic filter pipelines is not trivially achieved. But we believe we have identified some key challenges.

4 USING ILP FOR “COMPLEX” APPLICATIONS
It has been suggested that ILP will not yield significant benefits in performance terms when relatively complex protocol processing is involved. We have studied a picture server application. The application executes an Application Level Framed transport protocol in user space (Chrisment, 1994). Essentially, this permits the entire processing of each Minimum Coded Unit (MCU, which is a block of 16 x 16 pixels) at the client. The ordering of MCUs is not important, and each MCU can be decompressed and displayed on the screen on reception. The application uses JPEG compression (Hamilton, 1992), with the essential encoding/decoding and display steps derived from the XV software (Bradley, 1993).

We have carried out a thorough analysis of the client (receiver) code. The data manipulation steps are presented in Table 3 (for a fuller description of the processing steps, and of the instrumentation, the reader is referred to our workshop paper (Fry and Ghosh, 1995)). We note that most of these processing steps are generic protocol functions, which may be found individually and/or in various combinations in other application protocol stacks.

<table>
<thead>
<tr>
<th>Protocol Processing Step</th>
<th>Data Unit</th>
<th>Data Movements</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy to kernel</td>
<td>MAC Packet</td>
<td>1xR/W</td>
</tr>
<tr>
<td>UDP checksum verification</td>
<td>UDP Packet</td>
<td>1xR/W</td>
</tr>
<tr>
<td>copy to application space</td>
<td>ADU</td>
<td>1xR/W</td>
</tr>
<tr>
<td>Huffman decode**</td>
<td>6 DCT Blocks</td>
<td>1xW, 1xR/W</td>
</tr>
<tr>
<td>reverse DCT*</td>
<td>6 DCT Blocks</td>
<td>4xR/W</td>
</tr>
<tr>
<td>expand**</td>
<td>Macro Block (24)</td>
<td>1xR/W</td>
</tr>
<tr>
<td>ycc to rgb*</td>
<td>Macro Block (24)</td>
<td>2xR/W</td>
</tr>
<tr>
<td>dither for 8 bit display**</td>
<td>Macro Block (8)</td>
<td>2xR/W</td>
</tr>
<tr>
<td>output x image</td>
<td>X Image</td>
<td>1xR/W</td>
</tr>
</tbody>
</table>

Each step involves at least one physical read/write of the data. Those steps marked * also transform the data, while steps with ** transform and alter the width of the data unit. The data format for the output of each stage is described informally in Column 2. Column 3 summarises the number of reads and writes of the data at each stage. In the Huffman decompression and reverse DCT stages the amount of data actually read and written is variable, i.e. the quantity of input and output is data dependent (as is the amount of processing). Some of the reads/writes are simply copying from one buffer to another without data processing. These steps are potentially redundant, and capable of optimisation. In summary: there are a total of seventeen reads and fifteen writes of data, performed in fifteen loops. Six stages perform some processing on the data.

4.1 Performance
We have measured the average processing time on a Sun Sparc 10 for each MCU of a typical “talking head” CIF image. We have broken down these measurements into processing steps, some of which aggregate steps from Table 3. Our initial results are shown in Table 4.

<table>
<thead>
<tr>
<th>Protocol Step</th>
<th>Time (us)</th>
<th>Percentage</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP Checksum</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>other system (copy, csw)</td>
<td>270</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>pre DCT</td>
<td>553</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>DCT</td>
<td>458</td>
<td>11</td>
<td>30</td>
</tr>
<tr>
<td>post DCT</td>
<td>2166</td>
<td>51</td>
<td>81</td>
</tr>
<tr>
<td>output to display</td>
<td>804</td>
<td>19</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>4257</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

It is immediately apparent that the majority of processing time is occurring after the DCT stage. This is explained by the data expansion that occurs. Our compressed image — which is received from the network — consists of a total of 49,664 bytes. This is expanded to a CIF 24 bit colour image, which is a total of 304,128 bytes = an expansion factor of over six times. The full expansion has occurred at the end of the DCT stage. The cost of checksum processing and cross-domain copying is minor, correlating with the size of the (compressed) data at that stage.

So, while it has commonly been held that the intensive computation involved in compression and decompression dominates image and video applications, our results indicate that data manipulations performed on uncompressed data are just as significant. Furthermore, with the advent of modern processor architectures such as the DEC Alpha, the cost of complex computation vis-a-vis data movement is a widely discussed issue.

A further experiment suggests that, even for computationally intensive processing stages like reverse DCT, the costs of data movement and iteration are comparable to the costs of computation (Fry and Ghosh, 1995). A similar impact of memory bandwidth on video has been reported elsewhere (Patel, Smith and Rowe, 1993).
4.2 Some Problems With Filters
A simple approach to integration is to perform all layers of protocol processing in a single “for loop”, representing a single read-process-write cycle. In the above case, there are some major transformations performed on the data at certain stages, notably the huffman decode and reverse DCT. Intuitively, these transformations present problems for a simple “for loop” style of integration, and for filters. These transformations change the width of the data. All previous work on ILP and filters has assumed fixed and equal size inputs and outputs for each filter.

Furthermore parts of the processing pipeline, such as the DCT component, make non sequential accesses into an input array. In the highly optimised DCT implementation used for this application the accesses are a function of the 8x8 input array. So in the case of the DCT, we would ideally like the input and output to be 64 words wide rather than one word, as is the case with word filters. Alternatively, such a filter could buffer 64 words of state before its processing is triggered.

We are investigating whether such a datum width is compatible with ILP, how filters can accommodate such widely varying widths between input and output, and how performance is affected under various cache scenarios.

6 CONCLUSIONS
We have begun to explore how modular ALF/ILP systems can be constructed, suggesting that the ideas of smart buffers and filter pipelines may be integrated in a flexible and extensible manner.

We have performed experiments and drawn initial conclusions on the requirements and effectiveness of pipelines that are both modular and dynamically adaptable. Our broad conclusion is that there appear to be trade-offs between generality and performance. Efficient interface mechanisms between dynamic pipeline components are also difficult to realise. More work is required to develop efficient control mechanisms.

We have also produced evidence that ALF/ILP will be useful for distributed multimedia systems, since data movement and iteration is still the dominant overhead in many cases. However the presentation processing stages for complex compression/decompression present significant challenges for the pipeline approach.

We will conduct further research using the smart buffer and filter pipeline approaches. Such filters (and their interfaces) will be capable of re-use. They will be the building blocks for a tool that will perform automatic synthesis of ILP communication modules to suit the needs of different applications.

6 REFERENCES

BIOGRAPHICAL NOTES
Michael Fry received his PhD from the University of Sydney. He worked for a number of years in the computer industry designing and constructing real-time monitoring systems. He is currently an Associate Professor with the School of Computing Sciences at the University of Technology, Sydney. His active research interests include distributed multimedia systems, high performance protocol architectures, and nomadic computing.

Atanu Ghosh is a Research Fellow in the Department of Computer Science, University College London. He has a BSc(Eng) from Queen Mary College, University of London 1983 and a MSc in Data Communication Networks and Distributed Systems from University College London 1991. He has worked in industry on operating systems and networking. His current research interest is exploiting Application Level Framing concepts for the next generation of networked applications.
Towards Predictable ILP Performance — Controlling Communication Buffer Cache Effects*

B Ahlgren
Swedish Institute of Computer Science,
Box 1263, S-164 28 Kista, Sweden
E-mail: Bengt.Ahlgren@sics.se.

M Björkman
P Gunningberg
Uppsala University,
Department of Computer Systems,
Box 325, S-751 05 Uppsala, Sweden
E-mail: Mats.Bjorkman@docs.uu.se, Per.Gunningberg@docs.uu.se.

Cache memory behaviour is becoming more and more important as the speed of CPUs is increasing faster than the speed of memories. The operation of caches are statistical which means that the system level performance becomes unpredictable.

In this paper we investigate the worst case behaviour of cache line conflicts in the context of communication protocols implemented using Integrated Layer Processing. The goal of our work is to control the cache by placing communication buffers and code in non-conflicting positions in the cache. The result would be higher and more predictable performance. Our first results indicate that the worst case behaviour can be up to almost four times slower than the best case.

*Supported by the Swedish National Board for Industrial and Technical Development, under Esprit Basic Research Action project HIPPARC EC-AUS004.

Copyright © 1996, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the ACJ's copyright notice is given and that reference is made to the publication, its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

1 INTRODUCTION
The bandwidth of the primary memory system is more and more becoming the processing bottleneck of modern computers and in particular the bottleneck for their communica
tion performance (Druschel, Abbott, Pagels and Peterson, 1993; Smith and Traw, 1993). Since the speed of CPUs is increasing at a faster rate than the latency of memory systems is decreasing, the bottleneck is becoming more severe (Patterson and Hennessy, 1994; Wulf and McKee, 1995).

Modern CPU technology depends on cache memories for delivering high computation performance. Caches give a statistical increase in performance, but the gain is achieved at the price of less predictability. The speed at which applications run on such systems is therefore very much influenced on how well the applications use the CPU cache, both the instruction cache and the data cache.

The unpredictability comes in part from conflicts between different items in the cache competing for the same cache line. For communication protocols, the items are communication data buffers, other data structures used by the communication code path, the executed code and the processor stack. If it is possible to control conflicts between these, the resulting performance will be higher and more predictable.

It should be noted that the results presented from our experiments to control cache conflicts are applicable to other applications exhibiting similar behaviour as communication.

Integrated Layer Processing (ILP) (Clark and Tennenhouse, 1990) is a protocol implementation technique which reduces the number of data accesses to memory. ILP integrates the data manipulation functions of several protocol layers into one routine. Examples of data manipulation functions are presentation encoding and checksum calculation. The functions are run in succession on a small quantity of data, typically a word or a couple of words. Intermediate results between functions need not be written out to memory, but can instead be kept in registers or in cache. In the best case, data need to be read from main memory only once and written only once. With ILP, the transfer of communication data in and out of memory is thus minimised, which relieves the memory bottleneck and therefore has a potential of producing better performance in the common case where memory access time is the bottleneck.

The probability of finding a data item in the cache depends on spatial and temporal locality of reference. Communication data, by nature, exhibit poor temporal locality since new data is continuously sent or received. In an ILP environment where memory accesses are minimised the temporal locality is even less, since, in the ideal case, data will be read only once and written at most once. It is therefore no need to save a cache line with communication data for the future once it has been processed.

Communication data has, however, a higher degree of spatial locality since data are stored sequentially in data buffers, and since ILP loops exhibit a very regular behaviour,
reading and writing the buffers sequentially. There is a potential gain here in prefetching data to avoid cache misses. To conclude, the benefit from caching in our context is from the increased speed of subsequent sequential accesses to data in the same cache line, not from repetitive accesses to the same memory location.

1.1 Hypothesis
It is well known that employing CPU caches generally makes performance of computer systems less predictable, resulting in unpredictable end system communication performance. Our hypothesis is that, with suitable cache management, it is possible to obtain more predictable communication performance, with the additional benefit of increased performance because of fewer cache conflicts.

By treating the cache much in the same way as the virtual memory system or the CPU registers, it should be possible to avoid cache conflicts. This means allocation of parts of the cache for certain data and code. The data buffers, code, stacks and other data structures should be address aligned in such a way that they do not compete for the same cache line. Ideally, all data and code for the ILP loop would fit in the cache without conflicts. Then, only the first reference to each cache line would cause a load from main memory. This miss can often be hidden by using prefetching as mentioned above.

Note that the type of conflicts we are investigating are conflicts which almost only depend on the local behaviour of the communication code path and the data structures it references.

1.2 Approach
We are performing an experimental study to evaluate the hypothesis that predictability can be increased by managing the data cache. Interesting questions about the hypothesis are: When do cache conflicts exist? How much does a conflict cost? How likely is a conflict?

In this paper we present results from experiments evaluating the worst case behaviour of conflicts. The experiments are micro-benchmarks aimed at exploring the detailed behaviour of the situations causing cache conflicts.

2 RELATED WORK
Cache behaviour has been the focus of intensive studies for the last decade. The increased gap between processor speed and memory speed has further accentuated the need for systems with good cache hit rates. While instruction caching has drawn most researchers' attention, several researchers have investigated the possibilities to control data placement in the cache in order to avoid cache conflicts (Bershad and Chen, 1994; Kessler and Hill, 1992).

In the context of communication, however, not much work has been done on cache behaviour. University of Arizona has given special attention to caching for communication, both on the data side (Pagels, Druschel and Peterson, 1993) and on the instruction side (Mosberger, Peterson and O'Malley, 1995). For the instruction side, they investigate in how to use compiler techniques to increase cache performance. Salehi, Kurose and Towsey (1994) have studied cache behaviour in the context of parallelised communication protocols.

The original ILP concept was proposed by Clark and Tennenhouse (1990). It has been successfully used to fold together checksum computation with the kernel to user copy in UNIX by, e.g., Partridge and Pink (1993). Gunningberg, Partridge, Sirokin and Victor (1991) have demonstrated how ordering and some header constraints could be handled in ILP loops with several functions. Abbott and Peterson (1993) presents more general technique to handle this integration. They also present results from performance measurements of the technique. More recent work on ILP has been reported within the HIPPARCH project. The earlier work as well as the newer HIPPARCH work have observed and commented on the importance of cache behaviour and location of communication data for the success of ILP. In an earlier paper (Ahlgren, Gunningberg and Moldeklev, 1995), we discussed how a minimal copy data buffer management system should be designed in the context of an ILP loop in order to avoid conflicts. Braun and Diot (1995) report on and compare cache hit rates of an ILP implementation and a corresponding non-ILP implementation.

3 CACHE ARCHITECTURES
There are several design parameters of a cache that affect the possibility for conflicts besides its size. A cache is organised in a number of lines. Each line can hold a number of sequential bytes from primary memory. A common line size is 32 bytes.

The cache can be virtually indexed or physically indexed. In a virtually indexed cache the virtual address of a data item is used to calculate the location of the item in the cache. This makes it possible for the application program to control the location of its data items in the cache by choosing an appropriate address. In a physically indexed cache, the physical address is used to calculate the location. In this case the application needs support from the operating system to be able to determine the physical address of a data item.

The cache can be direct mapped, which means that the low bits of the data item's address directly indexes the cache. If the cache is 64 KB, as on the SPARCstation 2, the low 16 bits are used. In these caches, any two data items with addresses having identical low bits index to the same line and therefore conflict. The alternative to direct mapped is varying levels of associativity. These caches do not suffer from conflicts in the same way, but the drawback is that they are slower and more expensive.

When the processor writes to memory, the cache can behave in the following ways. It can be write-through, meaning that a memory write operation is scheduled immediately, or write-back, meaning that the write is scheduled at some time in the future. The cache can also be write-allocate, which means that if a data item is not present in the cache when the
processor writes, a cache location is allocated to store the data for future references.

A processor can have separate (or split) caches for instructions and data. The alternative is a unified cache holding both instructions and data. The computer system can also have several levels of caches, where the first level is the smallest, fastest and closest to the CPU. A typical high performance processor of today has split first level caches and a fairly large (256 KB to 1 MB) unified second level cache. The first level caches can have some associativity, but the second level is usually direct mapped.

4 THE COST OF CONFLICTS

We have performed a set of experiments to find out how severe cache conflicts can be. For these experiments we used the SPARCstation 2. It has a 64 KByte unified data and instruction cache with 32 byte lines. The cache is virtually indexed and direct mapped. It is write-through and does not allocate on write. This cache design is simple which simplifies experimentation and performance analysis. But the design is a few years old and therefore is not completely representative for the state-of-the-art. It has, however, some characteristics in common with today’s designs. Most, if not all, of today’s systems have unified and direct mapped second level caches. Many systems, e.g., the DEC Alpha and the HP PA RISC, also have direct mapped first level caches. We therefore believe that our experiments have some relevance to these systems as well.

We implemented a measurement program whose core is a loop in which different code easily can be inserted. The loop runs once over an input buffer producing data in an output buffer. The program measures the time over single invocations of this loop using the standard gettimeofday() system call. On SunOS 4.1.x this call has a resolution of 1 µs. The results presented below are median values of 1,000 measurements corrected for measurement overhead. The median value is often only one or two microseconds larger than the minimum measured value. The maximum value can, however, be almost any number because of interrupts and scheduling of other processes. The mean value is therefore not interesting.

The program gives complete control over how the input and output buffers are allocated and located in the cache. The buffers can be located in non-conflicting locations, or in locations conflicting in all combinations of input, output and code. We have taken care to avoid cache conflicts with the operating system code executed by gettimeofday().

The program also controls the cache temperature of the buffers. If “warm input” is selected, the program reads the input buffer before the measurement loop to load the buffer into the cache. It can also flush a particular buffer to make the cache cold.

We have experimented with two different loops. One is a simple copy loop and the other is Abbott and Peterson’s bswap/ pes/cksum ILP loop (Abbott and Peterson, 1993).

4.1 Simple copy loop

The code in the simple copy loop is shown in Figure 1. The purpose of this loop is to show the extreme case when an ILP loop has very little processing.

```
L77158:
1d [k15], %63
dec %24
kst %24
st %63, [k15]
ic 4, %13
bg L77158
inc 4, %15
```

Figure 1: The simple copy loop: C source code left, and compiled SPARC instructions right.

The left four bars in Figure 2 show the performance when the buffers are not conflicting for different cache temperature combinations of the buffers. One interesting thing to note is that even though the cache is write-through, it is beneficial to have the output buffer in cache. (As a side note, the cache apparently aggregates the single word writes into double word bus transfers.)

The best performance when data is actually read from and written to memory, the “ci/wo” case, is 85.9 ns/byte or 93.1 Mbit/s. This is about 75% of the maximum copy performance of this machine (about 64 ns/byte or 125 Mbit/s). It is not difficult to get closer to the maximum by slightly unrolling the loop and using the SPARC double

![Graph showing cache status and performance](image)
word load and store instructions. This loop exhibits similar behaviour as the simple loop but is a little less sensitive to input/output conflict.

"Wi/co" and "ci/co" are however more interesting, because they are the common cases in communication.

The two bars to the right in Figure 2 show the situation when the input and output buffers conflict in the cache. In the "warm input" case, each word located at the beginning of a cache line is read from the cache. All other reads result in cache misses. The interesting thing to note is that although the cache does not allocate on write, it seems to invalidate any other data at that cache location.

When we compare the conflicting cases with the non-conflicting we see that when the cache is cold (ci/co) the difference is a factor of 2.4, and when the cache is warm (wi/wo) the difference is a factor of 3.7. In the other for communication interesting case (wi/co) the difference is a factor of 2.7.

Figure 3 show the performance degradation when the input buffer conflicts with the executed code. The loop code is seven instructions or 28 bytes long and distributed over two cache lines. So it is only these two cache lines (64 bytes) which conflict with the input buffer. The measurements are done with different buffer sizes ranging from 2 KB to 128 bytes. With a 2 KB buffer the degradation is small which is expected since only 1/32 of the buffer conflicts. For a 128 byte buffer the degradation is large (3-4 times) because half of the buffer conflicts.

The code fragment which conflicts in our experiment is very small. In a situation with more complex computation or an unrolled loop, the code occupies a substantially larger space which results in more conflicts.

4.2 BSWAP/PES/CKSUM loop

The ILP loop used by Abbott and Peterson (1993) (see Figure 4) combines three operations: byte swap, PES (a pseudo encryption operation) and a checksum.

Figure 5 shows the per byte cost of the BSWAP/PES/CKSUM loop. The left four bars are without cache conflicts for different cache temperatures. The two bars to the right show the performance when the input and output buffers conflict in the cache. Compared to the simple copy loop, the processing time of the BSWAP/PES/CKSUM loop is about twice in the non-conflict case, but there is only a small difference in the conflict case. This indicates that the no conflict case is compute bound, while the conflict case is memory bound.

The performance degradation when the input and output buffers conflict is about 55% when the cache is warm, and about 30% when the cache is cold.

Figure 6 shows the performance effects when the input buffer conflicts with the executed loop code. We see a similar behaviour as the simple loop with a larger degradation for smaller buffers. The difference is that we get a peak for the 256 byte buffers and then a lower degradation for 128 byte buffers. This is due to the code size of BSWAP/PES/CKSUM loop is six lines or 192 bytes. The 128 byte buffer does not conflict with all of the code.

```c
for (; i > 0; i--) { /* READ A WORD OF INPUT */
    DataWord = *inputBuffer++;
    /* BSWAP */
    DataWord = ((DataWord & 0x0F0F0F0F) << 8) |
               ((DataWord & 0xFF0F0F0F) >> 8);
    /* PES */
    if (!awaitingSecondWord) { /* DataWord is first word of a pair */
        firstWord = DataWord;
        awaitingSecondWord = TRUE;
    } else { /* DataWord is second word of a pair */
        secondWord = DataWord;
        awaitingSecondWord = FALSE;
        DataWord = (firstWord & 0x0F0F0F0F) | (secondWord & 0x0F0F0F0F);
        /* CKSUM */
        sum += (DataWord & 0xFFFF) + (DataWord >> 16);
    /* WRITE A WORD OF OUTPUT */
    *outputBuffer++ = DataWord;
    DataWord = (firstWord & 0x0F0F0F0F) | (secondWord & 0x0F0F0F0F);
        /* CKSUM */
        sum += (DataWord & 0xFFFF) + (DataWord >> 16);
    }
    /* WRITE A WORD OF OUTPUT */
    *outputBuffer++ = DataWord;
}
```

Figure 4: Abbott and Peterson’s BSWAP/PES/CKSUM loop.
There are also two additional more or less expected conclusions. When the computation in the ILP loop increases, conflict between the input and output buffers has lesser impact. With careful prefetching, the impact of the conflict probably can be completely hidden. But the increasing performance gap between processors and memory is continuously making it more difficult. More computation in the loop also means larger code size, which increases the possibility of conflicts between the code and the buffers.

Further work includes measuring the cache behaviour on more platforms such as the SuperSPARC, the HP PA RISC and the DEC Alpha. We also plan to study the cache behaviour of different ILP loops using the SimICS simulator (Magnusson, 1995). This simulator executes real code and can be extended with different cache architectures. It can provide detailed information of the cache behaviour of the executed code.

REFERENCES


AHLGREN, B., GUNNINGBERG, P. and MOLDEKLEV, K. (1995): Increasing communication performance with a minimal-copy data path supporting ILP and ALF. Accepted for publication in a special issue of Journal of High Speed Networks on the HIPPARCH workshop.


ERRATA

School of Electrical Engineering,
University of Technology, Sydney
21 March 1996

Dear Editor,

We wish to correct our paper Structured Graph: A Visual Formalism for Scalable Graph Based CASE Tools by Mark Sifer and John Potter, which appeared in the February 1996 issue of the Australian Computer Journal.

Typographical errors arose in the definition of full flows (Definition 3.3) and lower bounds (Definition B.5) which were not detected in the galley proofs. In the definition $prods_I$, all occurrences of the term $cons_I$ should have been underlines to restrict the term to minimal consumers, in line with the preceding discussion. Here is a correct version of the definition:

Definition 3.3 Full Flows

Full flow producer for leaf links: $prods_I : Link \to \emptyset$ Node
For given $prods$ and $cons$, and $I : links$

$prods_I = \uparrow (prods_I - X) - X$

where,

$X = (cons_I) \cup (cons_I)^I$

Full flow consumer for leaf links: $cons_I : Link \to \emptyset$ Node
is defined analogously.

In the definition of lower bounds the less than or equal sign should have been reversed and the lower case $s$ on the LHS should have appeared in upper case. Here is a correct version:

$S^f = \{ m : P | (\forall s \in S) s \geq m \}$

Yours faithfully,

Mark Sifer

THE AUSTRALIAN COMPUTER JOURNAL, VOL. 28, No 2, MAY 1996
BOOK REVIEWS


Whilst Kulkarni provides a good introduction to the application of neural networks to image understanding, I found it an even better introduction to image understanding generally ('image understanding' as defined by the author comprises both image processing and pattern recognition).

Kulkarni's rationale, as stated in the introductory chapter, is: 'ANN models are preferred for image understanding tasks because of their parallel processing capabilities as well as learning and decision making abilities.' Nevertheless, I feel the author pushed this line a little too strongly by incorporating a chapter on ANN (parallel?) hardware.

The book comprises 10 chapters overall, these being:

1. Introduction (ANN models; image understanding)
2. Preprocessing (grey scale manipulation; edge enhancement; noise removal)
3. Feature extraction
4. Textual Analysis
5. Supervised Classifiers (probability density functions; linear discriminators; backpropagation)
6. Unsupervised Classifiers (k-means clustering; competitive learning, ART, Kohonen)
7. Associative Memories (BAM; Selective Reflex Memory; Hopfield; Counterpropagation)
8. 3D Structures and Motion
9. Neurocomputing
10. Applications

Most chapters provide a good summary of the fundamental concepts, backed up by a list of the key references.

Not surprisingly, Kulkarni leans heavily on his own research papers, as evidenced by the following list of ANN models which appear in the book:

- Grossberg & Todorovic (brightness perception)
- Carpenter et al (CORT-X filter for boundary segmentation)
- Dhawan & Dufrener (self-organising net for grey level restoration)
- Kulkarni (feedback-forward net for image restoration)
- Zhou et al (Hopfield net for image restoration)
- Kulkarni & Byars (ANN model for Fourier/Radon Transform feature extraction)
- Widow et al (ADALINE for invariant feature extraction)
- Kulkarni & Byars (ANN model for texture analysis)
- Kulkarni et al (backpropagation)
- Kulkarni & Whitson (competitive learning)
- Kulkarni & Yazdanpanah (BAM)
- Wechsler & Zimmerman (distributed AM)
- Zhou & Chehappa (Hopfield net for optical flow; motion description)
- Zhu et al (Hopfield net for motion estimation)

The book concludes with brief descriptions of eight image understanding applications, namely:

- remote sensing
- medical image processing
- fingerprints
- character recognition
- characterisation of faces
- data compression
- knowledge-based pattern recognition
- extraction of weak targets in high clutter environments

The only real problem I have with Kulkarni's book is the publisher's asking price of AS110!


NEILSEN, J. (1995): Multimedia and Hypertext: the Internet and Beyond, AP Professional, 480pp., price unstated

What an interesting read this has been! For those people, and I include myself amongst them, who have had minimal exposure to hypertext except, of course, via the World Wide Web (WWW) and its associated browsers, this book will be quite an eye-opener in terms of discussing what hypertext systems can be, should be and possibly will be in the future. I for one, and I readily admit that I am a newcomer in this area, did not realise that there were so many issues involved, nor that there were such a variety of methods for constructing hypertext systems. Jakob Nielsen, who has been involved in this field for a number of years, has done a good job of discussing the issues involved and describing existing hypertext systems although, at times, one tended to get the feeling that he was discussing technology for technology's sake.

Turning now to the contents of the book, there are thirteen chapters in all, plus a bibliography and a very comprehensive index. Chapters 1 and 2 define hypertext, multimedia and hypermedia and illustrate this with an example of a hypertext system. The author, prefers to use the "traditional" term hypertext to refer to all systems whether or not they include just text or text plus multimedia, seeing no reason to treat text-only systems differently. Chapter 3 gives an overview of the history of hypertext systems from Vannevar Bush's Memex system of 1945, through Englebarth's NLS demonstration of 1968, the first Macintosh systems of 1984, HyperCard (1987), Mosaic and the World Wide Web to the present day.

In order to gain a feel for the issues involved in the design and production of hypertext systems, chapter 4 describes a variety actual hypertext systems. Having set the scene, the next two chapters consider the architecture of hypertext systems and the hardware needed for such systems. Since the first exposure many people will have to hypertext systems will be through the WWW it is hardly surprising that there is a chapter devoted to hypertext on the Internet, HTML and the Mosaic style interfaces. Chapters 8 to 10 are probably the most academic in the book covering such issues as information retrieval and overload, navigation of large hypertexts, and usability of hypertext systems. Authoring usable and informative hypertext systems is a complex matter and chapter 11 briefly examines interfaces for authoring, toolkits, cooperative authoring and the way hypertext alters the author's ability to determine how readers should be introduced to the information contained in the document. This problem of relinquishing autonomy to the reader is highlighted in chapter 12 where two examples are given of how existing textual documents were converted to hypertext. Chapter 13 engages in some star gazing, looking at the future of hypertext, just what you make of this will depend largely on how much you feel technology drives changes in society.

This is an academic book. It has been thoroughly researched both from the point of printed literature and existing commercial and research hypertext systems. Given its academic slant, it is still quite readable and the annotated bibliography of both text and internet sources (amounting to approximately 80 pages) is far beyond that of some other recent offerings I have reviewed. However, I must admit to wishing that I had access to some of the systems described so that I could try out particular features for myself — the prose describes so that I could try out particular features for myself — the prose is well illustrated with clear, high quality and informative graphics. Highly recommended for all readers with more than a passing interest in the subject area.

Andrew Wray
Victoria University of Technology (Footscray)


This book is not nearly as interesting as its title would suggest. The book is in fact the 'Proceedings of the 2nd International Workshop on the Numerical Solution of Markov Chains'. Nevertheless I found the book interesting reading.

Overall I was impressed by the quality of the papers in this workshop. The papers are well written and interesting. The 37 papers cover a diverse range of topics. Not being an active researcher in the area I am unsure what would be of particular interest to my students. However, the preface identifies the most interesting developments being in stability and conditioning, Krylov subspace based methods for transient solutions, procedures for matrix geometric problems with quadratic convergence, analysis of the qth algorithm, and advances in the use of stochastic automata networks for modelling.

A. Bloesch
University of Queensland

THE AUSTRALIAN COMPUTER JOURNAL, VOL. 28, No 2, MAY 1996
Dr Edwards believes the development of internet technology will never put libraries at risk of becoming obsolete, but instead give them a new and more challenging role.

"Libraries will always provide information, but instead of just being used locally, new technology allows more global access", she said.

Dr Edwards said with the increasingly widespread use of the internet and other computing technologies, students needed to have a better grounding in the use of computers while at school.

"It would make life easier for universities if students were given a better computer grounding in schools. But unfortunately we still need to overcome the problem of a limited number of school teachers who are well trained in computing", Dr Edwards said.

However, regardless of the increasing need for computers, Dr Edwards believes the technology will never replace face-to-face communication.

"Employers need someone who can work in a team environment, talk to users and analyse problems rather than someone who just has good skills", she said.

That's the greatest misconception about computing sciences. Most students who enrol think it's all about maths, but employers want communication abilities".

"Technology on its own does not solve problems, people do".

OLTC and EDNA: OPEN LEARNING — ON LINE

On Line Open Learning is a learner-driven approach to education and training which takes into account the aspirations, needs and capacities of individual learners — be they situated in educational institutions, workplaces, communities or homes. The current development of enabling information and communication technologies are producing some exciting open learning initiatives, including that of Education Network Australia (EdNA).

A co-operative initiative of the Commonwealth, State and Territory Governments, EdNA is a national, interactive on-line network service which will utilise existing and emerging information and communication technologies to provide an accessible, affordable and flexible education resource for all sectors of Australian education and training.

The Open Learning Technology Corporation Limited (OLTC) was established in 1993 by Australia’s Ministers for Education and Training to support, promote and facilitate open learning techniques and technologies, and is the governing body of EdNA.

At the LETA 96 Media Launch to be held 29 September to 4 October 1996 at the Adelaide Convention Centre, OLTC will discuss EdNA as a practical application of current and emerging information and communication technologies to the fields of education and training.

POWERHOUSE MUSEUM INFORMATION TECHNOLOGY CENTRE

Primary and secondary school students will soon experience leading-edge technologies such as digital photography, remote sensing and computer-aided design in the Powerhouse Museum Information Technology Centre (ITC) in Sydney which opened in March.

According to the Director of the Powerhouse Museum, Terence Measham, "the ITC continues the Powerhouse’s commitment to creating a greater understanding of the practical applications of science and technology. It is designed to give students visiting the Powerhouse Museum the opportunity to explore and develop their information technology skills".

"The ITC is divided into two sections: the IBM-sponsored Information Technology Classroom, and the Information Technology Applications Room. The two rooms are inter-networked and can be used separately or in combination", Mr Measham explained.

In the IBM IT classroom, students are introduced to new information technologies and can build their competency with a range of computer hardware and software. This includes digital cameras and desktop publishing software with which students have the opportunity to produce their own digital newspaper.

Managing Director of IBM Australia, Mr Doug Elix said the IBM classroom would help provide students with a good general education as well as a specific skills — both of which are required by employers today.

"We are delighted to be able to contribute to the development of an educated Australian workforce through this sponsorship. I believe ventures of this sort have a real potential to add to the future competitiveness of Australian business", Mr Elix said.

The Applications Room gives students access to a range of current information technology applications for industry, business, education, science and leisure.

Students can read and monitor the news from around the world, manufacture products on computer controlled lathes and use digital manipulation software to retouch and manipulate photographs. Engineering drawings, product designs, building plans and sewing patterns can be produced and modified using Computer Aided Design (CAD) software.

From the beginner to the advanced student, the ITC caters for a range of computer knowledge and skills. Primary and high school students studying in a variety of learning areas including the visual arts, science, design and technology, English and computer studies will find the ITC a valuable resource.

There are a number of ITC packages available for school groups and throughout the year the Powerhouse Museum will hold a series of special events, open days and courses for the general public. Bookings are essential and can be made by calling (02) 217 0222.

APPLE SUPPORTS WORLD RECOGNISED COMPUTER ARTIST

Internationally recognised Melbourne artist, Diane Mantzaris, celebrates more than 10 years of using Apple Macintosh personal computers by announcing a new sponsorship deal with Apple Computer Victoria.
Mantzaris is highly regarded around the world as being a pioneer and key figure in the utilisation of computing technology as a component of contemporary Australian art production.

Since her work on an early model Apple Macintosh personal computer, she has continued to push the medium through her creative use of computer-generated imagery.

With the help of Apple Victoria, Mantzaris will push artistic boundaries even further. The company has provided her with an Apple Power Macintosh 8100 with which she can continue creating her unique style of computer-generated images.

According to Mr Simon Froude, Victorian State Manager for Apple Computer Australia, Diane Mantzaris, the artist, reflects the Apple philosophy in that they are “both highly creative, innovative and ahead of their time”.

People who are knowledgeable about computers and are used to seeing the work that can be produced with today’s technology are also surprised at the depth of power and emotion that Diane’s work communicates: “And, they are even more astonished to learn that she created some of her most acclaimed early works on paper with an early model Apple Macintosh, with its six-inch screen and printed to an Apple ImageWriter”.

Diane and Apple Macintosh have travelled a long way together since those early days in the mid-eighties, and in a way, our support in this instance is very appropriate because it is reflective of the pioneering work carried out by both parties,” Mr Froude said.

Naturally, Mantzaris is keen to start producing more exciting work on her new Apple Power Macintosh personal computer.

Says Mantzaris: “I appreciate and welcome Apple Victoria’s involvement in the arts. The marriage of art and technology is sure to prosper when the designers of leading-edge technology work with artists in a practical and mutually beneficial way”.

DEAKIN UNIVERSITY CAN NOW ACCESS RVIB LIBRARY SERVICES THROUGH THE INTERNET

Deakin University campus at Warrnambool can boast an Australian first — it now offers on-line computer access to the Royal Victorian Institute for the Blind Library Services through the Internet.

Now Deakin Print Disability Service staff will be able to search catalogues for existing titles, order copies and check on production for books in Braille, audio or large print formats at the touch of a button. They are able to access the RVIB Library Services computer management system directly.

Deakin co-ordinates the production of textbooks and other study materials with the RVIB, for the 51 students with disabilities or learning difficulties it serves throughout the University, on and off-campus. Some 540 documents will be produced during 1996 alone.

Previously, information took time to gather and many phone calls had to be made. The new on-line service, facilitated by VICNET, takes only minutes to process this information.

Deakin initiated this pilot project through DEET Equity funding. It ensures swift and cost-effective delivery of course material for students for whom early access is a critical factor in determining academic success.

Once established, a similar service is proposed for other tertiary institutions who are also supplying alternative format materials to students across Victoria.

MULTIMEDIA DEVELOPMENT FUND

The Victorian Government’s recently established Victoria 21 Multimedia Development Fund has announced the first round of approvals for funding.

Two million dollars per annum is being managed by the newly established Victoria 21 Multimedia Development Fund which is open to application by private sector organisations with projects that will create Australian content in the areas of entertainment, art, culture, education, design, business and information and which demonstrate clear commercial potential.

Ricci Swart, the Manager of the Fund, believes that the Fund will encourage a thriving multimedia industry in Victoria.

“We are delighted with the first round of approvals for funding. Two projects have received funding. In addition, Electric Alchemy received a Producer Package of $322,250 to assist with overheads and market attendance costs associated with three projects in development. We have a huge pool of creative talent in this State and with the establishment of this Fund, the Victorian government has enabled us to continue to maintain our lead in communications and multimedia development,” she said.

Projects at the Multimedia Education Unit of the University of Melbourne. She has considerable experience in video and multimedia production, “and is a recipient of national and international educational media awards.

Monthly evaluation rounds will take place before a committee of industry practitioners and announcements on successful projects will be made regularly.

ASC INTERNET HOME PAGE LAUNCHED

The Australian Securities Commission (ASC) has launched its first home page on the Internet as part of its commitment to improve community access to company information.

The home page is designed to provide small to medium businesses and private investors with information on a wide range of services offered by the ASC. Some of the major beneficiaries of this new home page will be people who live outside the major metropolitan areas who haven’t known how to get immediate access to the ASC’s information.

ASC Chairman, Alan Cameron, announced the ASC home page in conjunction with the launch of the ASC Infoline (1300 300 630) and said the combination of these two innovations now made the ASC more accessible to all small businesses throughout Australia.

“With a growing number of investors in Australia and continuing community concern about corporate misconduct, there is a need to provide better access to company information”, Mr Cameron said.

“There is a growing number of small businesses, individual investors and families who have personal computers that are connected to the ‘net’.”

“The information contained on this new service will help people who operate small companies gain access to ASC information while they are at home or at work.”

The page includes information about how the ASC works, how to report company misconduct, information on starting, running and winding up companies, changes to the Corporations Law, contact names, addresses and direct telephone numbers for key ASC staff, links to related areas such as consumer protection and overseas securities regulatory authorities.

The ASC home page is one of a number of initiatives aimed at making it easier for people to tap into the wealth of corporate information held by the ASC.

The ASC home page address is http://www.asc.gov.au.

DDPP IMPRESSES LEADING TECHNOLOGY SUPPLIERS

The launch of Digital Publishing Design Print, in conjunction with the 1996 Adobe Expo and Design Conference, has been received exceptionally well by suppliers to the graphic communications industry.

Within days of the announcement, Peter Blackey — Sales Manager of organisers Reed Exhibition Companies (Australia), had received over 50 enquiries from some of the biggest names in these industries.

Starting from such a strong position, DDPP is set to become the Asia-Pacific region’s benchmark for all that is new in the field of printing and publishing using both paper and screen-based visually rich information.

DDPP, held in conjunction with the Adobe Expo and Design Conference, offers a complete learning and commercial experience for industry visitors. Educational workshops and guided technology tours are main features in the exhibition area whilst on the conference side, Adobe Systems will be seeking to exceed their previous three annual events’ very high standards for international and Australian experts presenting a broad variety of learning experiences to delegates.

This is a combined event that definitely should not be missed by anyone in printing, publishing, on-line services, graphic design, photography, multi-media, CD-ROM production and Internet/World Wide Web services.

DDPP takes place in conjunction with the Adobe Expo and Design Conference from Thursday 12 September to Saturday 14 September 1996, at the Sydney Convention & Exhibition Centre, Darling Harbour.