Lightweight Protocols for Distributed Systems

Thesis for Submission for PhD

J.Crowcroft

Department of Computer Science
University College London, Gower Street, London WC1E 6BT.

ABSTRACT

A methodology and architecture have been developed that contrast sharply with common interpretations of the Open Systems Interconnection concept of Layering. The thesis is that while many functionalities are required to support the wide range of services in a Distributed System, no more than two concurrent layers of protocol are required below the application. The rest of the functions required may be specified and implemented as in-line code, or procedure calls. The proliferation of similar functions such as multiplexing and error recovery, through more than one or two layers in the OSI model, contradicts many basic rules of modularisation, and leads to inherent inefficiency and lack of robustness. Furthermore, it makes the implementation of Open Systems in hardware almost intractable. A key design issue is how to construct the automatic adaptation mechanisms that will allow a protocol to match the service requirements of the user.

The arguments in this thesis are based on the experience of the design and implementation of three major classes of protocol for distributed systems: a transaction protocol (ESP), a reliable multicast protocol (MSP), and a Stream Protocol (TCP). A number of performance measurements of these protocols, including the latter, the DoD Transmission Control Protocol, were carried out to support the thesis.

Finally, the impact of these protocols and this architecture on the switching fabric are investigated. We examine mechanisms for congestion control and fair sharing of the bandwidth in a large internet connected by datagram routers and MAC Bridges, with a variety of link technologies ranging from terrestrial to satellite. The discussion of Interconnection techniques is included because it is the view of the author that understanding traffic distribution algorithms is vital to understanding end to end protocol performance.

Acknowledgements

I would like to thank everyone in my family who encouraged me to finally do some work. [1]

Thanks are also due to Ben Bacarisse, Mark Riddoch, Karen Paliwoda, Steve Easterbrook, members of the SATNET[2] measurements Task Force, Peter Kirstein, Soren Sorenson, Peter Lloyd and Robert Cole for assistance at various points during this research.[3]

Thesis: Aims and Goals

The aim of this research was to develop protocols that are designed and engineered for scalable distributed systems support. To this end, it was a goal to produce an efficient transport protocol for one-to-one and one-to-many Remote Procedure Call support that can also be used for conventional (connection-oriented) applications. An important goal is that we understand limits to the performance of such protocols in the Local and Wide Area.

Sub-goals included: possible hardware support for transport protocols; link and network level security implications of non-trivial size distributed systems; some understanding of possible formal specification of aspects of the protocols developed.

The work described in this thesis was carried out between 1985 and 1988. The author believes that despite the time elapsed since then and the presentation of this work, many of the findings are still highly relevant, especially the protocol designs and performance results.

---

1. Thanks especially to the members of the IAB's End-to-End Research Group for the alternative title - the Unbearable Lightness of Protocols (Craig Partridge after Milan Kundera).
2. This work was supported in part by DARPA under Contract Number N00014-86-0092. Any views expressed in this thesis are those of the author alone.
3. Some inspiration was drawn from the writings of Myles na Gopaleen concerning the Research Bureau in the Irish Times:

   Recently I referred briefly to a new type of telephone patented by the Research Bureau. It is designed to meet an urgent social requirement. Nearly everybody likes to have a telephone in the house, not so much for its utility (which is very dubious), but for the social standing it implies. A telephone on display in your house means that you have at least some 'friend' or 'friends' - that there is somebody in the world who thinks it worthwhile to communicate with you. It also suggests that you must 'keep in touch', that great business and municipal undertakings would collapse unless your advice could be obtained at a moments notice. With a telephone in your house you are 'important'. But a real telephone costs far too much and in any event would cause you endless annoyance. To remedy this stay tough affairs the Bureau has devised a selection of bogus telephones. They are entirely self contained, powered by dry batteries and where talk can be heard coming from the instrument, this is done by a tiny photoelectric mechanism. Each instrument is fitted, of course, with a bogus wire which may be embedded in the wainscoating. The cheapest model is simply a dud instrument. Nothing happens if you pick up the receiver and this model can only be used safely if you are certain that none of the visitors to you house will say "Do you mind if I use your phone?" The next model has this same drawback but it has the advantage that at a given hour every night it begins to ring. Buzz- buzz, buzz-buzz, buzz-buzz. You must be quick to get it before an obliging visitor 'helps' you. You pick up the receiver and say "Who? The Taoiseach? (Irish Prime Minister) Oh, very well. Put him on." The 'conversation' that follows is up to yourself. The more expensive instruments ring and speak and they are designed on the basis that a visitor will answer. In one model a voice says, "Is this the Hammond Lane Foundry?". Your visitor will say no automatically and the instrument will immediately ring off. In another model an urgent female voice says "New York calling So-and-So" - mentioning your name - "Is So-and-So there?" and keeps mechanically repeating this formula after the heartless manner of telephone girls. You spring to the instrument as quickly as possible and close the deal for the purchase of a half share in General Motors. These latter instruments are a bit risky as few householders can be relied upon to avoid fantastically exaggerated conversations. The safest of the lot is a Model 2B, which gives an engaged tone no matter what number is dialled by the innocent visitor. This is dead safe.

Contents: Organisation and Layout

The thesis is presented in 8 chapters. The core of the work is presented in chapters three, four and five. Here we present a **simple** transport protocol for RPC support, extensions for one-to-many conversations, and performance measurements of a similar protocol. This meets the primary aim of the thesis. The requirement for such protocols has long been identified, and references are given throughout the text in support of this.

The outline of all the chapters is as follows:

1. Distributed Systems Protocol Architectures.

   In this chapter, we review the services provided by the transmission network and those required by the user. We then review protocols in distributed systems, and finally present our Distributed Systems Protocol Architecture,

2. Protocol Implementation Design Methodology

   In this chapter we review Protocol Design Methodologies, and examine relative layering strategies and overheads. We compare the layering overheads versus flexibility and examine Finite State Machine Specifications and direct implementation from them. We look at mapping End to End protocols to Bounded Buffers

3. Sequential Exchange Protocol

   In this chapter we present the design of the Sequential Exchange Protocol, a Transaction Protocol for RPC. We look at the need for Presentation Level Integration, the need to support recursive calls, and then look beyond simple RPC at Messaging and Streaming.

   This protocol is an alternative to VMTP[Che86a] and can be used as a suitable improvement on the psuedo transport protocol used on UDP for support of NFS/Sun RPC.[Inc86a, Mic89a] It is intended primarily as a transaction protocol for RPC, something which has been identified as missing for a decade. We look at how there is a need for Presentation Level Integration. The protocol also supports recursive calls. This chapter also examines what is needed beyond RPC such as Messaging/Streaming used for supporting networked window systems.

4. Multicast.

   This chapter presents the design of a multicast protocol, which is an enhancement to the transaction protocol presented in Chapter 3. The aim of the protocol is to provide a service equivalent to a sequence of reliable sequential unicasts between a client and a number of servers, whilst using the broadcast nature of some networks to reduce both the order of the number of packets transmitted and the overall time needed to collect replies. Appendix 1 introduces a Distributed Conferencing Program that can make use of multicast, and Appendix 2 outlines possible OSI protocol support for the windowing system used in the conferencing application.

5. Protocol Performance.

   This chapter describes the tools and measurement mechanisms used to investigate the performance of the Transmission Control Protocol (TCP) over the Atlantic Packet Satellite Network. One of the main tools used was developed by Stephen Easterbrook at UCL, while the other was provided by Van Jacobson of LBL.

   The relevance of this work is that similar dynamic mechanisms to those used by TCP are also employed in the design of both the Sequential Exchange Protocol and the Multicast protocol described in the previous two chapters. It might be argued that TCP has largely been deployed successfully in workstations on LANs. However, historically it has been designed to function over a wider range of underlying media than any other protocol, and to this end is widely adaptable. The SATNET was a network at just one extreme of the operating parameters for WANs.

We take a pragmatic look at retransmission schemes, flow control schemes over Datagram Networks, Congestion Control Issues and Adaptation in Protocols.

Appendix 6 describes some further details of transport protocol implementation and specification techniques. We look into problems of Concurrency, Robustness and Efficiency, Presentation Interaction, and Buffer Ownership.[ISO83a] Appendix 8 covers possible protocol hardware support, including parallel processing, pros and cons of Front-End Processors, Support for rate control, support for sequencing/resequencing and support for packetisation and presentation parsing.

As transmission speeds increase, more stress is being placed on protocol implementations. Here we explore which salient functions can be moved into, or supported by, specialist hardware. Since we are arguing for a 2 or 3 process implementation of the communications stack, parallel processing has limited scope in any given conversation. However, for large hosts, there may be many conversations in progress simultaneously. There are also a number of specialist tasks that may be done such as in-line presentation encoding/decoding, checksumming and encryption/decryption. These could all be done in a pipeline of processors. In intermediate systems in the network, there are possible benefits in using a processor per interface.

6. Managing the Interconnection of LANs

   In this chapter we present some of the techniques for building extended LANs that have been used to build metropolitan scale distributed systems without needing Metropolitan Area Networks. It is important to understand the impact of these techniques on packet loss, reordering and duplication so that we can build robust transport protocols.

   This approach to scaling networks may seem to be *bottom up*, compared with the RACE programme approach in designing a pan-European network from scratch. It does have major advantages in being based in high speed technology that is available now, so that performance of real systems can be studied rather than merely modelled. We look at routing and performance problems.

   This chapter continues on to examine the interconnection of LANs with "MAC"[Ber85a] (or link/frame level)[IEE85a] Bridges, and Internetwork level relays ("Gateways" or "Routers" in DoD Terminology).[Hin83a, Bog80a] The following issues are mainly addressed in the context of the DoD Protocol suite and interconnected Ethernets: managing interconnected LANs, implications for Bridges, Routers and Relays of high performance, localisation of traffic versus the cost of routing and addressing protocols.

   Appendix 3 describes the application of these ideas in the context of a university or campus system configuration. Appendix 4 describes a possible distributed application of some of these ideas. Appendix 5 describes the configuration of the Alvey Admiral network, which was the testbed for for some of this research. Appendix 7 describes some further measurement experiments and results carried out on another such configuration, on the UCL Campus network.

7. LAN Access Control at the MAC level

   Chapter seven includes a description of some extended network security features that we have devised for extended LANs. One criticism often levelled at network architectures is that security is added as an afterthought. In the process of this work, we show that at least some levels of security (in the area of access control on a per host or subnetwork basis) can be provided at a reasonable level using fairly modest mechanisms.

8. Conclusions and Further Study

   We conclude by looking at the emergence of very fast Networks and Servers including the ISDN, B-ISDN and MANs, and the new requirement for real time packet switching.

In appendix 9, we survey some of the formal description and specification techniques that may be used to help design and implement protocols for distributed systems.[And91a, Hoa78a] Formal techniques for producing parallel distributed software are relatively new. Here we examine the applicability of some of the ideas to protocol implementation, and attempt to evolve some principles of safe programming. We look at Specification, Verification and Proof, and how transport services should be proved to map into Bounded Buffers. We also ask if behavioural equivalence is interesting, or is CSP more useful?

1. Chapter 1: Distributed Systems Protocol Architectures

In this chapter, we review the services provided by the wire and those required by the user. This simple three layer approach is an attempt to expose the basic parameters in the protocol design space. We avoid reproducing a description of the Open Systems Interconnection Architecture as this is well understood and described well elsewhere. In contrast, we review protocols designed for distributed systems from the research community during the 1980s.[IEE83a] Much of the material presented in this and the next chapter consists of descriptions of tools and building blocks which can be put together in a variety of ways. It is how these components go together in distributed systems now that may be different from the peer to peer communication models of traditional communications.

When considering why we need specialised communications software, and how it will differ from existing applications and systems software, we must consider the hierarchy, consisting of:

- What the *User* wants to communicate.
- What the *Operating System* provides for Communication Support.
- What the *Wire* provides in terms of basic communication.

1.1 Users

By users, we mean, ultimately, human computer-users, but also include a wide range of application programs. The kind of things users wish to communicate cover a wide spectrum:

Traditional	Contemporary	Modern
Analogue Voice	Terminal Access	Window Based Graphics Terminal
Telex	File Transfer (lumpy or whole)	Shared Files/File Access
TV	Electronic Mail	Conferencing/Multimedia Mail
Mail Order	Remote Job Entry	Distributed Execution
		Load Sharing/Replicated Services

TABLE 1. Range of Network Uses

1.2 The Operating System

The operating system provides generic software for communication, whereas the user provides anything more specific to the task in hand. **Generic** communication software is partly what this thesis is all about.

1.3 The Wire

The *Wire* is in reality a wide range of services, ranging from a *modem* with an acoustic coupler, attached to a conventional telephone, right through to a High Speed digital packet switched network. This is illustrated in Table 1 and Figure 2.

The range Wide Area (National and International) networks extends a long way as exemplified in Table 2:

	Phone	Trunks	Packet Switched
Numbers	600 Million	10,000s	100s
Technology	Modems/PCs	Modem/Muxes	DTEs/Gateways
Usage	French Minitel	Banks etc	Research

TABLE 2. Range of Network Technology

1.4 Switching Methodologies

In switched digital networks, the switching nodes can operate in three different ways (refer also to Figure 1):

- **Circuit Switching**

  The switches on a route establish a physical circuit between two hosts for the duration of the communication session. This is how the telephone system operates.

- **Message Switching**

  The sending host parcels up data into long messages which are transferred step by step over each link between the switches and delivered to the recipient.

- **Packet Switching**

  As message switching, but the parcels are short, so that the switching nodes can store numbers of them in memory.

Packet switching is favoured for computer communication: It makes good use of the links. Between bursts of data between one pair of hosts. other hosts may use the same links and switches. Delays due to storing short packets in switches are short compared to storing long messages. Switches need less memory/buffer space to hold packets for forwarding than for messages. Hardware to implement packet switches is now relatively cheap. Switches and links may break during a communication session, but alternative routes can be found. Packet switched networks have been in use for over 25 years. Early examples are the US ARPANET and the UK National Physical Laboratory Network. The ARPANET evolved into the Internet, which by the end of 1992 connected well over 1 million hosts worldwide. At the same time, the PTTs provide an international packet switched network. In the UK this is BT's PSS (Packet Switched Stream) Service.

Packet switching requires that packets contain addresses. For a packet to get from one host to another through a series of switches, each switch on the route requires to know where the packet is going. Thus as we move from circuit switching to packet switching, we move intelligence into the network, and we move the information for that intelligence to act on into control information in packet headers. A constant debate throughout the history of packet switched networks has been just how much intelligence there should be in the network, and just how much state held within network switches between one packet and the next for a given conversation. At one extreme, exemplified by the Internet there is little state but there is a great deal of intelligence concerned with dynamic routing. At the other extreme, exemplified by the Public Data Networks, and emulating the physical circuit (hence *virtual circuit*), there is a great deal of state setup before communication between pairs of end-points. This allows for simpler packet formats, and for the possibility of hop-by-hop flow control.

There has been much recent renewal of the debate concerning the efficacy of hop-by-hop flow control versus end-to-end flow control as a bandwidth sharing and flow quarantining mechanisms. Although this is largely beyond the scope of this thesis, it is touched on in chapter 5.

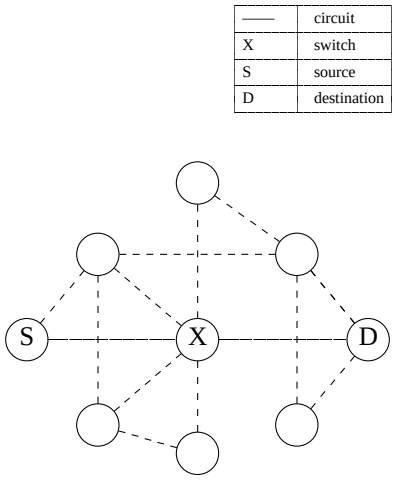


Figure 1. Switched Network

The range of network characteristics is wide:

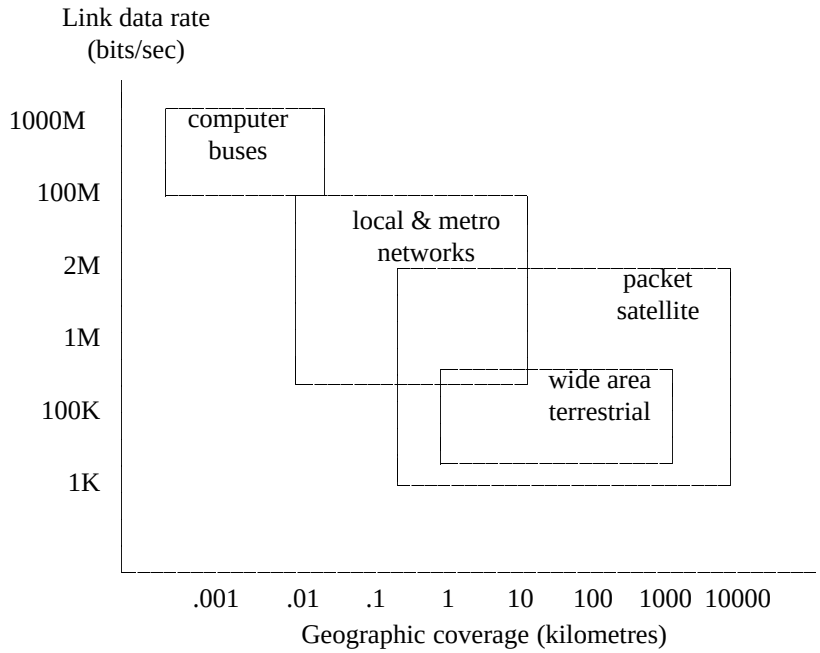


Figure 2. The Spectrum of WAN and LAN Characteristics

In Wide Area Networks (WAN), data rates range from 300 bps up to 45 Mbps. Error rates are on the order of 1 in 10^7. Delays vary from 100 msecs to seconds. We usually find bit serial interfaces. In Local Area Networks (LAN) data rates range from 10 Kbps to 1000 Mbps. Error rates are on the order of 1 in 10^9. Delays are of order msecs. We typically find frame or byte level interfaces to Local Area Networks.

1.5 Packets on the network

Packets are placed on the network. What is their fate? Even if the network is essentially a reliable bit pipe, it is possible for it to break, independent of the end hosts communicating. Practice shows that it is in fact far more likely to break than, for instance, the terminal interface device on a mainframe or a disk controller. If the network is packet switched, then other things can go wrong too in a variety of subtle ways.

Thus, depending on the type of network, a variety of disasters can befall a packet: The network may not know how to get this packet to its correct destination; the packet can just get lost due to lack of memory or a temporary fault in some component; a packet can be corrupted by electrical (or other) noise on a wire, or by faulty software; packets can be delivered to the remote end in a different order than they were sent; the same packet could be delivered multiple times to a destination; packets could be delivered by the network to a destination host faster than the host can deal with them; packets may even arrive that were not actually sent by anyone (invented by the network); the network may support many different optimal packet sizes at different places.

The consequences are many: There must be mechanisms for hosts to indicate where they wish to communicate - just as file systems allow us to indicate which file we want to read or write; there may be a single wire leading out of a host, which then splits in some way to go to multiple destinations. This means we need to have some kind of *addressing* mechanism, so that we can *multiplex* communication between many hosts on a network; the user needs some level of **reliability**. This can vary between some statistical level - digital voice or video may only require some percentage of data per second to be delivered from which a reasonable result can be constructed by the receiver. [For instance, humans only need 30% of the sound of someone's voice to be able to understand 99% of what is being said]; the user may need to restrict the rate at which data arrives. *real-Time* applications like voice and video require fairly exact rates of arrival. Print servers may support much lower data rates than typical modern networks. The user needs to be able to subdivide ("fragment" or "segment") and re-assemble the data into appropriate packet sizes. This may further involve a mechanism for discovering **Moga** the appropriate size, or rely on intermediate systems performing this function. It is inefficient to do this in too many places.**Ben82a**

The network must also be used responsibly and must protect itself against malicious use. Since it is often one of the more expensive resources an organisation can possess, it must also be used efficiently. In summary then, a Packet Switched Network is analogous to a huge statistical multiplexor.

1.6 Operating System Facilities

Most O/S support is for some form of communication and sharing of devices, and so has some facilities for building protocols. However, conventional device support usually assumes that devices are accessed over a local bus, and so are almost error free and with delays similar to memory access.[4] Also, sharing of devices is achieved by access being *only* via a device driver, and is therefore easily achieved. The system safely separates users from each other, and protects devices from aberrant behaviour of users' processes by restricting access through *system calls*.

---

4. This view of operating systems has a Unix bias. One thing that we do not address in this thesis is "what are the consequences for the end-to-end arguments for parallel operating systems (e.g. symmetric and non-symmetric parallel unixes such as Convex, Pyramid, Solbourne)"? Just what is the "end" - a particular piece of global memory or a particular CPU cache line?.

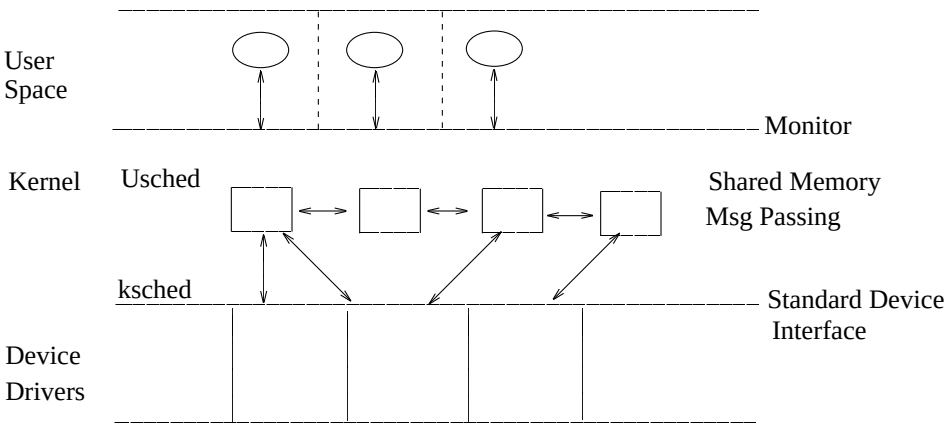


Figure 3. Operating System Structure - Process View:

Figure 3 illustrates the layering and process model of a typical uni-processor operating system of the 1980's such as 4.3BSD Unix.[Qua85a] Within the *Kernel*, or in a *Real Time System*, the structuring is rather different with closer cooperation between processes or threads of control, less protection, and the assumption that the programmer is more expert. Processes run in two main types of context (or address space) with memory management units controlled by the operating system to protect processes' memory from random, erroneous access from other processes. These are "User Space" and "Kernel Space' (in Unix terminology). There are then three levels of scheduling: Dispatching of interrupt service routines; Scheduling of kernel tasks (e.g. file system tasks, or network and transport protocol tasks); Scheduling of User Processes. More recently, this division of scheduling classes has gone further to include "threads", which operate within a single address space, within a single "user space" process. Systems such as Amoeba, MACH, Chorus and Windows-NT provide this abstraction.

1.7 Software for Intermediate Devices in a Network

In a packet switched network[Pou73a] , and in store and forward networks[Now78a] , we have special purposes boxes, variously called Bridges, Switches, Gateways and Relays. These will normally run real-time operating systems, and perform many of the same functions as general purpose end hosts, except for the upper levels which may not be required. There is often special hardware support to facilitate communications. e.g.:

- Intelligent Interfaces, including Content Addressable Memories, for de-multiplexing packets based on fields within a packet (addresses).
- Intelligent or Complex Buses (e.g. binary tree buses or even no buses).
- Specialised use of Memory Management Units (MMU).

Depending on the *architecture* of the network, and the choice of end host protocols, these intermediate nodes support different complexities of protocols, usually ranging[5] from **connectionless** to **connection oriented**.

1.8 Distributed Systems Protocol Architecture

The choice of where low level protocol functionality should go is dictated to a large extent by the distribution of problems and performance in the underlying interconnection fabric. Factors include where

5. Networks have been either connectionless service (CLNS) or connection oriented (CONS) until recently. Protocols for real time that have emerged such as ST (see chapter 11) have a connection setup and teardown, but little intermediate protocol state in the network; although resources are statistically set aside, they are not completely guaranteed.

the errors occur and where the delays are incurred, be it in transmission or queueing in bridges or routers. We argue that the current trend in technology has moved the engineering parameters towards connectionless networking, and pure end-to-end detection and recovery mechanisms.[6]

In the workstation arena, the Internet Protocol Architecture has arisen as the main structure for communications for distributed systems. In recent years, however, some shortcomings of this architecture have been pointed out. Clark and Tennenhouse propose Integrated Layer Processing in their paper[Cla90a] and Tennenhouse[Ten89a] again calls for a reduction in layers. Throughout this thesis we will explore the implications of these ideas in implementation.

Above an all pervasive Internetwork layer, traditional applications such as remote terminal access, file transfer and so forth have been built on the service provided by reliable stream protocols such as TCP.[Pos80a] This service has proved invaluable in simplifying the communications model for these applications, but has proved less appropriate to communication patterns involved in distributed computing. We now turn to the transport protocol requirements for these *users*.

1.9 Transactions

Cheriton[Che86a] describes a transaction protocol with multicast facilities tuned for page level file access, but also suitable for wider area distributed system transactions. Watson[Wat89a] describes a transaction protocol that ingeniously avoids problems with rapid system crash and reboots, using time changes rather than sequence or epoch numbers. Extending this work, Chesson[Inc90a] describes a protocol very similar to VMTP [Che86a] specially tuned for high performance interworking from graphics workstations to supercomputers. The protocol was also designed for VLSI implementation.

None of these transaction protocols support *Call Back* where the server of an RPC request (or descendants of the current call) may make a call back to the originating client, within the same *conversation*. This is despite the fact that this is a perfectly natural way for any program to behave (mutual recursion) whether distributed or not. Randell [Ran78a] describes this in detail in an early paper on the Newcastle Connection, the first distributed Unix with RPC.[Pan85a]

In chapter 3 we will describe a transaction transport protocol that supports callback. In chapter 4, we describe extensions to support multicast.

1.9.1 The Limit of Transactions

The sequential exchange paradigm leads to unacceptable latency for some applications.[Lis84a] It becomes necessary to provide a reliable message *streaming* facility (relaxing the strong synchronisation between the application interface send/receive pair of operations, and the corresponding exchange of messages with the server application. The X Protocol is an example of just such a system.[Get86a] VMTP and the Sequential Exchange Protocol both provide this functionality, although many other transaction protocols do not. Another approach is to provide fine grained efficient concurrency in servers and clients. This has become more widespread in systems recently.[Spe89a]

The protocols described in chapters 3 and 4 provide streaming of calls to improve application performance over that constrained by the latency of the network and turnaround time, when blocking awaiting the return from a conventional non-streamed conventional RPC.

1.10 Sequence and Time

Lamport[Lam78a] describes an algorithm for global agreement on sequencing. Based on these arguments Birman[Bir85a] describes *ABCAST*, a protocol providing totally ordered broadcast.

---

6. This trend is growing stronger as more and more of the networking infrastructure is provided by cabling. In the long term, pure optical switching will mean that we move to nearly error free transmission, while data rates will possibly outdistance processing speeds so far that software control in the network will have to be very lightweight indeed.

Mills [Mil85a] describes a protocol and algorithms for synchronising clocks in a distributed system. Liskov et al have combined Mills' work with Birman's[Wro90a] to describe an algorithm for minimising handshaking in messaging systems in a distributed system with synchronised clocks as described in Mills.

These mechanisms permit less hand shaking within distributed programs, since any ambiguity in ordering is removed by enforcing a global view of time, and thus of messages. There is a natural tradeoff between the extent to which this is enforced and the amount of concurrency in the system, but at the gain of significant simplicity for the programmer.[7]

1.11 Management

A Distributed System is not useful outside a research laboratory without management. In chapter 5 we look at performance management, and in appendix 8 suggest hardware support for performance enhancement. In chapter 6 and 7 we look at interconnection, routing management, and security. In appendix 9 we examine some formal aspects of a system that may in the future allow us to prove that we meet contractual obligations in terms of correctness, at least in terms of service.

1.12 Related work

There are many background references on Architectures[8]. For instance, Tanenbaum[Tan81a] and Halsall [Hal88a] describe the OSI and XNS architectures. See also the RFCs by Dave Clark and Comer[Com88a, Ste91a] for the DARPA Internetwork Architecture.

Distributed System Designers' views of relevance include:[Lam81a, Ada83a, Ben83a, Bog80a, Cel83a, Col83a] on Wide Area Communications and[COS84a, Cyp78a, DEC82a, ECM82a, Fri85a, Gif85a, Sch85a, IEE83a] on distributed systems and [Laz81a, Wec80a, Zim80a] on more general communication architectures. The architectures in these references are remarkably similar in that the layering and location of function or modularisation is largely similar; the majority are aligned with respect to choosing connectionless networking layer.

More recent work, for instance the thesis by Lixia Zhang on Architectures[Zha90a] where a rate based network architecture is outlined, and the XTP work[Che90a] merge more of the functions of the network and the transport layers, and also move more flow policing functionality into the network, while stopping short of distributing full connection state in switches.

---

7. Note that there is an implication for the security of a distributed system whose correct or performant operation depends on synchronised clocks. The synchronisation sub-system must be secure.

8. Webster's defines this as:

   ar.chi.tec.ture ´a:r-k*-.tek-ch*r n 1: the art or practice of designing and building structures and esp. habitable ones 2: formation or construction as or as if as the result of conscious act 3: architectural product or work 4: a method or style of building

2. Chapter 2: Protocol Implementation Design Methodology

In this chapter we review lower level Protocol Design Methodologies, and examine relative layering strategies and overheads. We describe some of the commonly accepted building blocks for protocols, ranging from the specification of formats for packets including header control information, and the *elements of procedure* or operational components of a protocol. We look at building blocks both for correct protocol operation, and for efficient provision of communication, both for the user and for the (usually shared) transmission network.

What is a **protocol**? First and foremost, it is a set of agreed rules on what is meant by each unit of data exchanged - i.e. it is the encoding or agreed meaning for bits/bytes/words exchanged. It is also a prescription of what procedures to adopt in each circumstance.[9] Examples of such rules for format include the use of English in international diplomacy, the use of particular dictionaries for scrabble and the use of the ascii code for characters stored in computers. In Computer Communications Networks, for reliability, flow control, resource sharing, etc, we need protocols with complex procedures. Some of these building blocks are described below.

2.1 Some agreed formats in protocols

The DoD Internet Datagram protocol is a format for a packet, used commonly on LANs (Ethernet) and WANs (the Internet). At the time of writing, the Internet now interconnects over one million host computers, and can be regarded as the most successful network, and employs one of the most successful protocol suites in existence. The IP Packet Format is show in Figure 4.

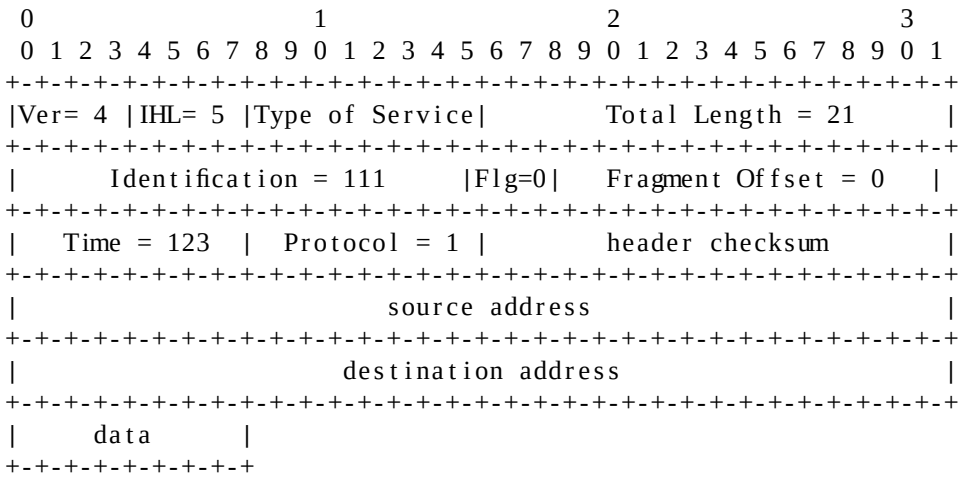


Figure 4. Example Structure of Internet Datagram

9. *"Nohow" - Tweedledee. "Contrariwise" - Tweedledum.* In Alice's Adventures in Wonderland, Charles Dodgson.

The order of bits and bytes must be specified unambiguously too, as shown in Figures 5 and 6:

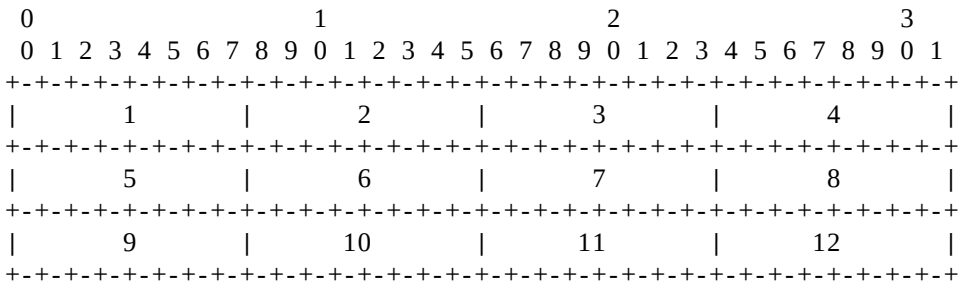


Figure 5. Transmission Order of Bytes

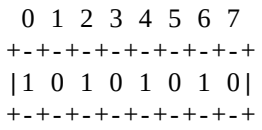


Figure 6. Significance of Bits

A long running (but pointless) debate over the merits of big-endian versus little-endian bytes and words is described well in Cohen.**Coh81a**

2.2 Some Rules for Protocols

Like a human conversation, a protocol is a set of agreed rules for structuring communication between parties. To make conversations possible, there must be some rules or protocols. Examples of such structures are: question and answer; the monologue (broadcast); the conversation; a cry for help (broadcast); an order and an acknowledgement; the use of announcement of identity of caller/callee in a telephone call; the agreement to abide by an arbiter in a cricket match; the handshake at the end of a business deal. There are many others.

Each end point of a communication session is *autonomous*, that is each can decide to stop or start *talking* at any time, or to interrupt another party. This involves *concurrency*, and is what makes the design and implementation of protocols difficult; the software to implement any protocol of significant interest involves concurrency. It is inherent in communication that there is true interleaving of events (programs managing each end of a conversation). There are non-deterministic events (users may type at a keyboard at any time, as someone on the telephone may speak at any moment). This means that polling implementations are not often feasible: the number of possible events that must be polled for and the time to execute a polling loop are prohibititive, and differing event streams happen at different, varying rates. This leads to event driven programming, and other ways of structuring communicating systems, often involving concurrency. We discuss this further below.

Computer Communications Protocols employ very limited patterns of exchange so that their design is tractable. There are a variety of graphical ways of depicting protocol behaviour:

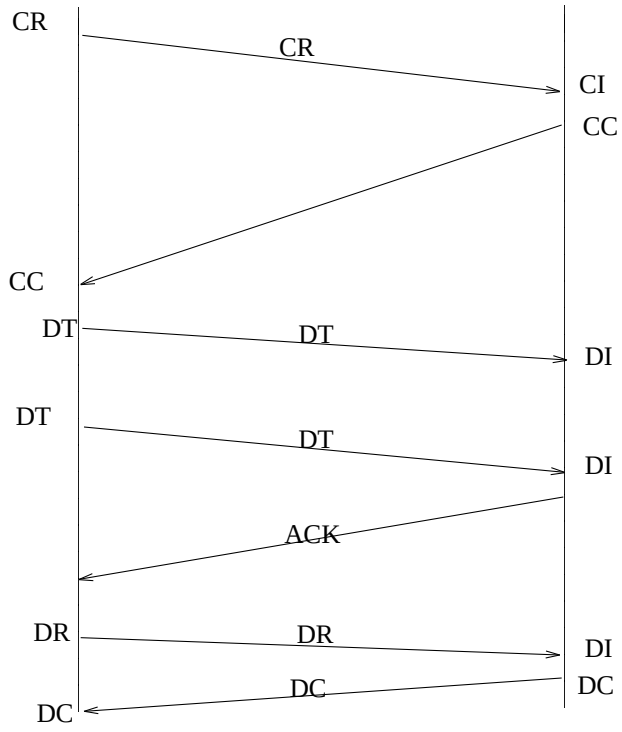


Figure 7. An example of an invented protocol behaviour

CR	Connection Request
CC	Connection Confirm
CI	Connection Indication
DT	Data
AK	Acknowledgement
DR	Disconnect Request
DC	Disconnect Confirm
DC	Disconnect Indication

TABLE 3. Packet Type and Event Key

Figure 7 and the Key in Table 3 exemplify a *Time-Sequence* diagram. Time increases down the page. The left and right vertical lines represent planes of interaction between layers (e.g. the network and a protocol) at each end. Sloping lines traversing the page represent packets traversing a transmission system, taking a finite (non-zero) amount of time. These lines are labelled with a packet type (e.g. Connection Request (CR)). The points where they start and end on vertical interfaces are labelled with event names (e.g Connection Indication (CI)).

2.3 What is Concurrency?

In the software world, we commonly find three types of concurrency:

1. Fake concurrency provided by a multiprogramming environment or operating system. This exists down to the level of *Interrupts* and up through *System Calls* to concurrent programming environments.
2. Real concurrency in a **tightly coupled** multi-processor environment. This does not interest us here, as communication in such an environment is generally provided by hardware, and such devices are special purpose (e.g. graph reduction machines for functional languages, Digital Array Processors etc).
3. Real concurrency in a **Loosely Coupled** Distributed system.

2.4 Interleaving

Because of the unpredictable nature of the real world input to a system - the *non-deterministic* order of events, it is far easier to model the system behaviour as a set of (cooperating) processes/tasks/threads, each of which deals with an appropriate stream of events. This can then be implemented using co-routines, or multi-processing. We then have to view the system *as if* these tasks actually run concurrently (at the same time).

What actually happens in a single system is that they are *interleaved*. This means that at any point in execution, one task could stop, then another run. There will be some indivisible (atomic) unit of execution that cannot be suspended. In *pre-emptive* operating systems, the atomic unit may be as small as a single instruction. In *Run to Completion* real-time systems, this unit will be decided by the programmer. In a distributed system, there is real concurrency whether we want it or not. This brings special problems - for instance *partial failure* of a system is possible.

2.5 Problems with Concurrency

The first and last cases of concurrent systems in the previous section concern us. What are the special programming problems in such systems?

Firstly, whether we have an interleaved system, or a real distributed system, we have to protect *Critical Regions* of code. These are sections of code that may alter the value of *Shared Variables*. Because of concurrency, it is possible for incorrect values to arise, if the sequence of operations on these variables is not carefully controlled. For instance, the arbitrary interleaving of two threads of control that both carry out a fetch, modify and store sequence of operations on the same variable results in a high probability that only one of the modifications is is made permanent. Secondly, we have to control ownership of resources. A number of incorrect behaviours can result from incorrect sequences of resource allocation, including deadlock, livelock and problems with fair allocation of resources.

Deadlock is illustrated in the classic case in Figure 8. Livelock is illustrated in Figure 9. Lack of Fairness is discussed further below.

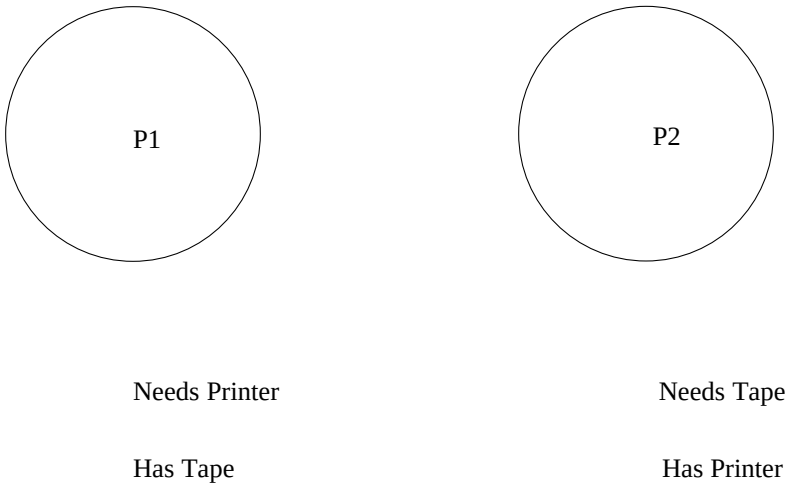


Figure 8. Deadlock:

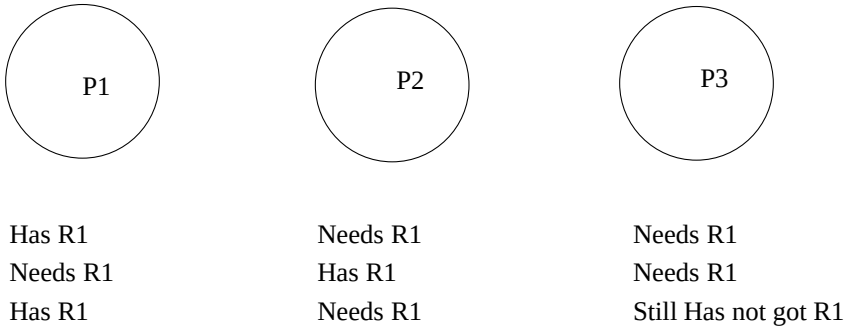


Figure 9. Livelock:

2.5.1 Fairness:

The presence of a resource greedy process can mean other processes do not get access to a resource as often as necessary. An example of this often occurs in operating systems which *over-prioritise* a tape drive, so that terminal access is radically slowed down when the tape is running.

Distributed systems involve another resource that must be shared fairly - this is the network. A packet switched network is effectively (logically speaking) *statistically multiplexed* between the various end hosts. Use of the network itself, as well as CPU and buffers and so on, must be controlled fairly. Fairness is complex enough in concurrent systems within operating systems.[Lam81a] When systems are autonomous to the extent that networked ones are, it becomes ever more difficult. Much of the work described in chapter 5 on TCP is in an attempt to get a better understanding of fair sharing of a network.

Andrews[And91a] defines 3 levels of fairness:

1. Unconditional Fairness

   Every unconditional atomic action (or task) is eventually scheduled.

   The networking analogy is that data is always eventually transferred, no matter how little the end user retries.

2. Weak Fairness

   Unconditionally fair and conditional atomic actions (tasks) are eventually executed if their guard becomes true and stays true. The networking analogy is that data is always eventually transferred, provided that once the end user has started to try to send data, they persist.

3. Strong Fairness

   Unconditionally fair and conditional atomic actions (tasks) are eventually executed if their guard is infinitely often true (i.e. true infinitely often in the history of a non-terminating task). The networking analogy is that data is always eventually transferred, so long as having started to try to send data, the end user persists, every now and then.

The model of windowing that TCP adopts (described in chapter 5) achieves a form of weak fair-sharing, and is in part a consequence of these considerations.

2.5.2 Formal Systems

Formal systems now exist for defining and checking the behaviour of such systems of distributed concurrent tasks - for instance Milner's CCS, Hoare's CSP, and the OSI LOTOS.

Hoare's CSP is based on techniques of program design and testing that derive from weakest pre-condition approaches, finding loop invariants and minimising *Interference* between concurrent modules, so that safe sequential programming techniques may be used as far as possible, and the combinatorial explosion of states of concurrent programs is slowed.

One application of these techniques has been the ISO layer approach for specifying protocols. [ISO84a] One of the dangerous consequences of this particular approach is that it is often *taken literally* in implementations. As we shall see, this can lead to very poor performance. A serious limitation of these approaches has been that although a formal specification can lead to the elimination of safety or liveness problems, the problems of timeliness and fairness are far harder to chase down. Indeed, it is often the case that hidden events are introduced to simplify a definition (by regarding these events as outside the scope of the system being specified). This very act can lead to lack of fairness.

2.6 Consumers, Producers and Critical Regions

Using the paradigm of layering for modularisation, we have a natural model for building communications protocols where a service is provided by a layer to a layer above. We have seen that there are only 2 external interfaces to a protocol layer - lower layer input events, and upper layer request events. Corresponding to these, there are requests from this layer to the layer below, and indications (events delivered to them) from this layer to the layer above. There are also internal events (e.g. timers) and control events (e.g. network resets, user aborts etc) We have modelled a layer as a set of cooperating tasks, processes or *threads of control*, which are essentially concurrent. Although this concurrency is usually fictional (i.e. interleaved execution by a single processor, rather than multiprocessor), it is convenient for designing and implementing protocols - a thread for each stream of events is easier to program than a calculated polling loop, with the fraction of time almost impossible to calculate for each event stream (to take just one poor alternative approach). Having chosen to implement most protocols as a set of cooperating concurrent threads of control, we are left with the problem that the cooperation is often by means of shared memory. This leads to one main problem, and a set of design rules:-

- Each thread within a layer (protocol) that deals with events from a layer above, is a consumer of those events, but a producer for events of the layer below (usually as a result).

- Each thread within a layer dealing with events from the layer below is a consumer of those events, and generally produces events for the layer above as a result.
- Thus through many layers we have a *chain* of producer from top down to the bottom, and another chain of producers from bottom to top. Each chain of producers is matched by a chain of consumers.
- All of this is to ensure lack of **deadlock**, although it cannot ensure fairness (lack of livelock etc).
- It is intuitively obvious that the system makes flow control reasonably easy to implement.

This software structure is illustrated in Figures 10 through 13. The flow of data and control through the system is shown in Figure 14.

```
-- 1/2 of Transport Protocol
--         Consumer of User events, Producer of Network requests...


        for(ever)
        {
                sleep(tsdu-q-event);

                prev = spl-ts()
                tsdu-pkt = deq(tsdu-q);
                spl-restore(prev)

                -- make up some tpdu (transport pkts)
                -- do all the rtx q work and set timers
                -- if this is that service/kind of transport protocol
                -- (and network service is not enough to give us the
                -- service we are asked for)

                prev = spl-ns()
                enqueue(nsdu-q, tpdu-pkt)
                spl-restore(prev)

                wakeup(nsdu-q-event);

        }
```

Figure 10. Transport Entity 1

```
-- 2/2 of Transport Protocol - TS2
--        Consumer of Network input events, Producer of User events

        for(ever)
        {
                sleep(tpdu-q-event);

                prev = spl-ts()
                tsdu-pkt = deq(tpdu-q);
                spl-previous(prev)

                -- decode tpdu
                -- queue for user
                -- generate acks (with appropriate window updates)

                prev = spl-ts()
                enqueue(tsdu-q, npdu-pkt)
                spl-restore(prev)

        }
```

Figure 11. Transport Entity 2

```
-- 1/2 of Network Protocol - NS1
--        Consumer of Transport events, Producer of Link requests

        for(ever)
        {
                sleep(nsdu-q-event);

                prev = spl-ns()
                tsdu-pkt = deq(nsdu-q);
                spl-previous(prev)

                -- make up some npdu (network pkts)
                -- do all the rtx q work and set timers
                -- if this is that kind of Network protocol

                prev = spl-ls()
                enqueue(lsdu-q, npdu-pkt)
                spl-restore(prev)

        }
```

Figure 12. Network Entity 1

```
-- 2/2 of Network Protocol
--         Consumer of  events, Producer of Network requests...


        for(ever)
        {
                sleep(npdu-q-event);

                prev = spl-ns()
                npdu-pkt = deq(npdu-q);
                spl-restore(prev)

                -- decode npdu, and demultiplex to appropriate
                -- transport entity

                prev = spl-ts()
                enqueue(tpdu-q, tpdu-pkt)
                spl-restore(prev)

                wakeup(tpdu-q-event);

        }
```

Figure 13. Network Entity 1

Some definitions:

- sleep - waits (suspends this task/process/thread) till some other task wakes us up with appropriate event).
- wakeup - wakeup *any* task sleeping for this event.
- spl-? - set priority level to *exclude* any tasks from waking up who need this level of access. This can be implemented using a semaphore, or else the entire protocol layer might be a monitor. spl-x - restore all previous priority levels.
- que - add a buffer/packet to a queue for another process to...
- dequeue - remove from a queue some buffer.
- -- - is a comment

Corresponding to Figure 10, is a consumer of network events (TS2) and producer of transport service events to the user. For Figure 12, there is a consumer of link events, producing network input to the this transport task (NS2).

Thus in a link plus transport protocol, for a full system from one host (user) to another, we have a chain of production and consumption which contains at least 8 tasks!

The mutual exclusion plus synchronisation prevents inconsistency or deadlock across the entire system. However, it is possible for a producer to produce more than a consumer can handle at any point in this system, as we have no model for flow control. Flow control of resources is usually achieved by counting the numbers of buffers being queued and dequeued at each producer/consumer boundary, and bounding this number (i.e. stopping production when we have queued some given number of events for a consumer who has not managed them yet).

Another aspect of communication which does not quite fit this paradigm as presented is that of fragmentation and reassembly. When sending a protocol data unit through a layer, it may be that a layer above has chosen a convenient unit size which is not convenient for the layer below. Some might construe this as poor design, but it is a common consequence of protocol implementation when based on a layered approach. This then requires the sender to break down N+1 layer Data Units into several N-layer PDUs. These must also be reassembled in a receiver or intermediate node. Which is the better choice is not clear. How to apply queueing, pacing or other techniques to fragments of a PDU is also not obvious. [10]

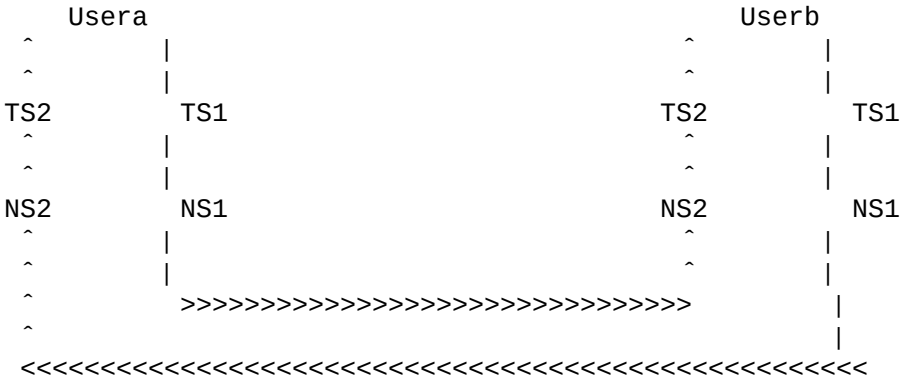


Figure 14. Flow of Information between Protocol Entities

The model presented above and in Figures 10 through 14 shows exactly why the number of concurrent layers implemented at each end should be minimised. Both the Sequenced Exchange Protocol and Multicast Transport Protocol were designed to run in such a system, with a maximum of 3 concurrent layers at each end. The cost in terms of latency and CPU usage associated with the context switch implied by the mutual exclusion above is only required for the network to transport level, and transport to application. Any other layer processing (e.g. presentation coding/decoding) should be achieved inline.

A connection-oriented network service requires at least one more process layer, and yet (as argued earlier) does not obviate the need for a powerful transport protocol.

2.7 The Internet Architecture

The DoD DARPA Internet Architecture**Lei85a** is an architecture based on two principles that differ in emphasis from the ISO OSI model:

- Fault Resilience built into the Network
- End-to-End *fate sharing* protocols.

The unit of interconnection is the *network*, rather than the single *link*. Fault resilience is achieved by choosing to support only the lowest common denominator of network services.

---

10. Of course it is possible that the flow of control packets (e.g. acknowledgements with piggybacked window information) in a reverse path may need to affect the flow of data in the forward direction. This would entail interlocks between the producer-consumer chains in each direction. This needs to be dealt with very carefully. Most protocols avoid this situation since they are not used in a genuine full duplex way (e..g one way bulk transfer, or request followed by response). Interactive application level protocols may have this problem, but have been limited in their resource requirements by being driven by a human (e.g. typing). In general, it may be best to avoid designing such protocols.

The Internet Datagram Protocol allows the interconnection of heterogeneous *networks* by datagram switching *gateways* (sub-level 3c in the OSI model), as illustrated in Figure 15. These can be relatively simple devices, and only need support enough of the functionality of the networks they connect to be able to send and receive datagrams on each network. The gateways that interconnect networks [Sun75a] can route datagrams dynamically depending on available paths and network loads.

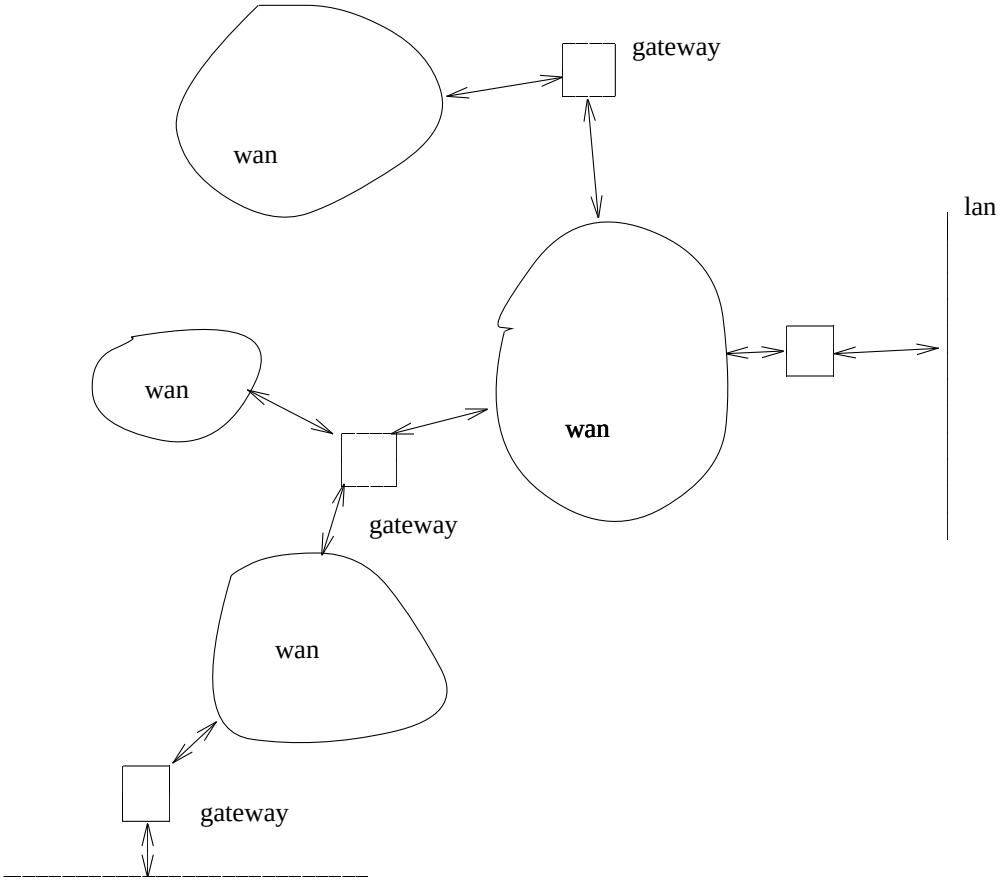


Figure 15. A Fictitious Internet

Datagrams support simple *best attempt* packet delivery.[Pos80a] This means that we need only to be able to send and receive packets, deal with addresses and checksum headers. Only error detection is done, and packets in error (either checksums fail, or mis-directed) are **discarded**. It is up to higher level protocols to deal with problems. Datagram protocols provide some further functions including fragmentation, to deal with networks that support varying maximum packet sizes;[Ben82a, Sho79a] time to live specification, to enable the network to discard packets too *old*; header checksums, to prevent corrupt packets unnecessarily congesting gateways.

There is usually **one** IP implementation (process/task/thread) per machine, and it is usually built in the kernel of the operating system. This may talk to several *subnet* or local network interfaces, which are usually implemented in drivers. Above the IP layer, there may be several transport protocols, each of which exists, once per machine. These must then *multiplex* between users/applications.

The main end-to-end protocol (*Transport Level* in ISO terminology) supports strong synchronisation of end hosts, reliability, ordered delivery and byte stream operation. The **Transmission Control Protocol** builds on top of **IP** to provide the underlying service which most users require for standard applications like terminal access and file transfer. This philosophy is well explained in Saltzer[Sal84a] and Clark.[Cla82a] TCP is more complex than most transport protocols, so that it can cope with the dynamic nature of the underlying

network service: the actual operating parameters can vary by orders of magnitude during a single *session*. For example, typical implementations can tolerate delay variations from 2 msecs to 20 secs, throughput in the range from 1bps to 20 Mbps and packet loss rates from 0 to 50%.

The Transmission Control Protocol[Pos80a] provides a reliable stream of bytes between two processes on different machines. It relies only on IP for packet delivery, and can deal with packets lost, out of order or duplicated. TCP provides the following protocol functionality:

- End-to-end addressing using *Ports*.
- Full Duplex connections, using a single packet format, with control and data indicated in a single 20 byte header.
- Error detection using low cost checksums.
- Recovery using an *Acknowledgement* and *Retransmission* scheme.
- Duplicate detection and segment ordering using sequence numbers.
- Flow control using *Dynamic Windowing*.[Cla82b]

2.7.1 Management

The Internet Architecture has a rich naming architecture with several levels.[Sho78a] The Address Resolution Protocol (ARP) is used for IP address to local network address mapping.[Plu82a] The ICMP is used for error reporting and diagnostics as well as subnet configuration request and response messages.[Pos80b] A plethora of routing information protocols including GGP/EGP/IGP/BGP is used for gateway to gateway exchanges to build up a distributed topological view.[Hin83a, Mil84a] The Domain Name Scheme is used for a service name (string) to IP address mapping.[Moc84a]

In contrast to management "in the large" that is found in the IEEE and ISO models of network management, these functions are embedded in the layers that their functions support. The SNMP protocol is used for monolithic network management access to Internet objects described in the Management Information Base objects in a remote system.[Cas90a] On the other hand, the embedded approach is exemplified by the inclusion of dynamic or adaptive protocols such as ARP for distributed address resolution and ICMP for route dissemination and traffic source quenching.

In chapters six and seven, this simplifying approach is used in looking at LAN interconnection and some possible embedded security mechanisms.

2.8 Protocol State and Parameterization

To deal with each connection, we must have a connection descriptor. This will contain adequate address information to distinguish a unique connection (source and destination IP host and TCP port), associated data and acknowledgement sequence numbers and transmit and receive window sizes. The connection descriptor must also hold timer information for each end pair. This is established in the first phase of the traditional connection setup, data transfer and disconnect steps of such a protocol, as shown in Figure 16. Timers and their relation to retransmissions as a result of lost packets (be they data packets or acknowledgements) are discussed further in Chapter 5. Figure 17 illustrates simple packet loss and the relation between a retransmit timer and round trip time.

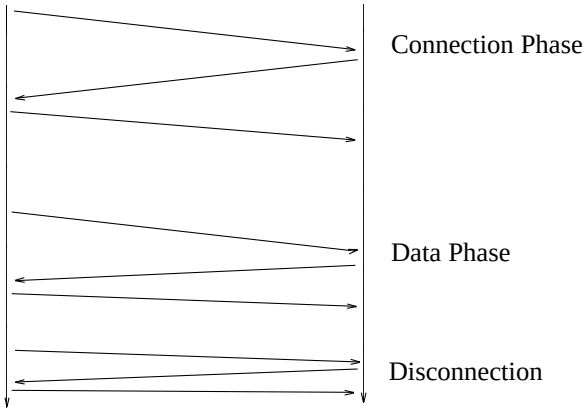


Figure 16. Connection Phases

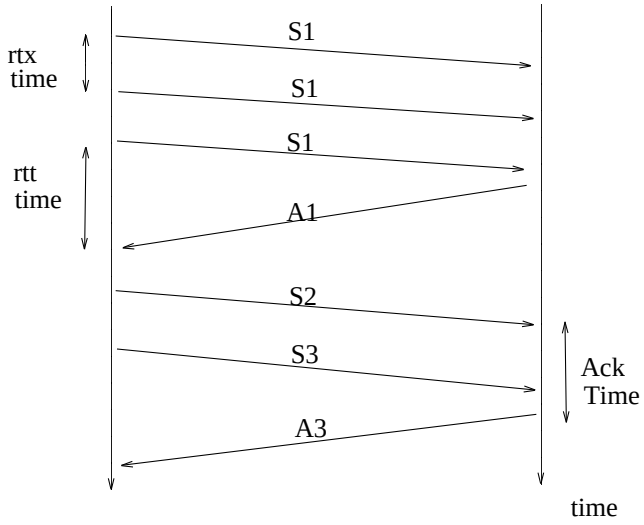


Figure 17. Timers and Bandwidth*Delay - Pipesize

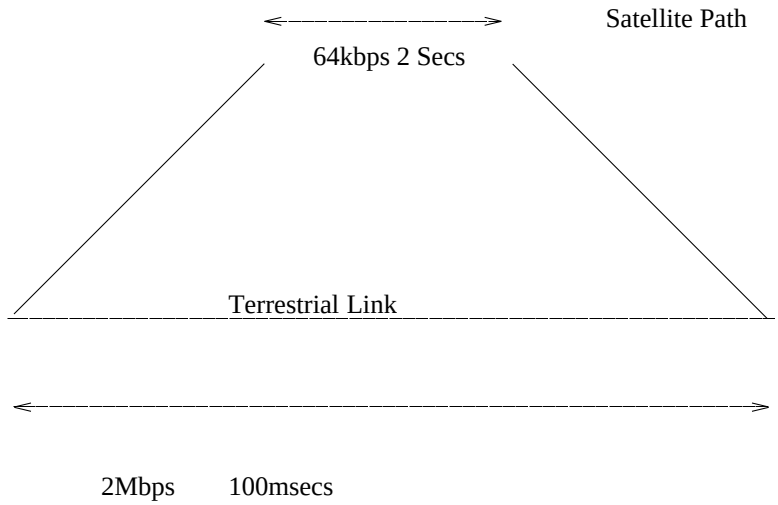


Figure 18. Windows and Delay/Bandwidth Product

The bandwidth/delay product in modern networks is such that to achieve reasonable throughput, *outstanding* packets must be permitted. This trend is increasing, mitigating against stop-and-go style protocols (at any level, be it transport, or application or elsewhere). Windows may used to specify a receive buffering, and with or without explicit "Receiver Ready/Receiver Not Ready" style messages, can *flow control* a sender. However, a window is **required** to make good use of modern networks. Consider two typical paths with the characteristics shown in figure 18, with a terrestrial megastream path and a satellite kilostream path, both supporting 1000 byte packets. A "stop and go" protocol would lead to possible throughput:

Terrestrial: 1000bytes in 100 msecs = 80Kbps.
Satellite: 1000bytes in 2 secs = 4Kbps.

The solution is to use dynamic windowing. The reason static windows are not sufficient in a datagram network is that the path may change. This is studied further in chapter five.

2.9 Different Architectures

The OSI and DoD architectures are designed specifically to support traditional applications such as Remote Terminal Access, File Transfer, Electronic Mail and Remote Job Entry. Modern applications in *Distributed Systems* require a different range of protocols. For instance, File Access can be built with a general facility for remote or distributed execution. Applications like this have different communication requirements which can be characterised as requiring the following features:

- Request/Response call oriented
- Low latency per call
- Calls possibly streamed
- Programming Language integration

We discuss below approaches that have emerged to provide this additional functionality.

2.9.1 Berkeley/Sun and First Order Distributed Computing

In the early 1980s, DARPA funded the Department of Electronic and Electrical Engineering at the University of California, Berkeley, to implement the Internet Protocols. The implementation team went further and provided general a purpose communications programming environment based on sockets, which is in some sense a *First Order* distributed computing system.

In 4.2/4.3 BSD Unix, they added some functionality to the standard communication functions: A simple *presentation* level: most machines fall into 4 classes: 16 bit/32 bit, big endian/little endian (as defined in

Cohen's paper);[Coh81a] rlogin (remote login), rsh (remote job entry), rcp (remote copy) provide *authentication*, though only as much as Unix understands, and assume a flat *user-id* space across a network; rexec (remote execution) on some systems provides a simple remote execution facility *from within programs*;

In the mid 1980s, researchers from Stanford took the next generation of ideas from research work at Xerox PARC such as courier[Cor81a] and set up Sun Microsystems ("Sun" originally stood for Stanford University Network). Their workstation systems provided a lot more functionality. A more complex presentation level, based on that in the Xerox Network System architecture, called *XDR*, eXterior Data Representation [Cor81a] was developed. This gave a higher level of language integration of communications, which we may refer to as *Second Order* distributed computing, since it permits programs to be first class objects in a network.[11]

Using this, a number of more sophisticated distributed applications emerged. The **Network Disk**, ND, provides *stateless* transparent remote raw disk access between different machines within the operating systems. **Network File System**, NFS, provides *stateless* transparent file access across machines. **Yellow Pages** is a distributed in-core database for name and location services, mapping user to permissions, name to address and so on. These were **all** implemented using **Remote Procedure Call**, or **RPC**.

The programmer writes normal C (or Pascal) programs that call procedures in the usual way. These procedures can actually reside in a different process, and that process on another machine. RPC then provides a mechanism for linking the *Client* program that calls the *server* together. At compile time, the two ends are linked with *stubs* - for the client this is a dummy set of the procedures, for the server, a dummy caller. These *stubs* are commonly generated automatically from the XDR specification of the *server* routines. At runtime, these dummy routines package up call and reply parameters supplied by client and server, and call some underlying protocol, to ship them over the network to the other process.

NFS and Yellow Pages are implemented using RPC. Programmer can write their own servers using the system too. Under the Alvey Admiral Project, UCL designed its own RPC system [Bac86a] with more language integration based on Birrell and Nelson's classic approach to RPC.[Bir84a] we next turn to transport protocols to support this higher level functionality.

2.9.2 VMTP and ESP

Underlying protocol support for RPC type systems is weak in general: Usually implementors have chosen to use simple exchange protocols that have a limit of one packets worth of call data, and one of reply data. They usually use a simplistic retransmission scheme to gain some level of reliability. They usually only work on a local area network, because of their low tolerance for errors.

Some (transport) protocols are appearing to support request/response protocols (closely matching the modern requirements mentioned earlier):

- Low latency
- Support for arbitrary size call and reply parameters.
- flow/rate control schemes to avoid over-running receivers.
- Support for more sophisticated retransmission schemes, allowing for variation in networks.
- Support for *multicast* calls to support applications like replicated databases,

Examples are Sequenced Exchange Protocol and VMTP.[Bac8.a, Che86a] VMTP is the protocol used to carry out distributed operations from the V-Kernel [Che86b] operating system. It is a request-response protocol,

---

11. The late 1980s and early 1990s have seen the appearance of what might be referred to as *Third Order* distributed computing, where systems can create new objects in a distributed way, through more powerful object oriented distributed language constructs. It is yet to be seen if these are successful.

with local network multicast support, and one unique feature - call forwarding. This allows a server to forward a call to a *better* server, rather than replying itself. The definition of "better" and its determination are decided by the application.

2.9.3 Other Systems

Other operating systems are now appearing with this kind of support: MACH (and the small XINU system) has kernel and user space support for exchange protocols; Chorus is very similar to MACH; System V.3 Unix has *stream* support for many kinds of protocol.

Most of these systems support **shared memory** or **message passing** between processes and the operating system, to reduce overheads of communication. *Context switching* between processes and the system is also made as low cost as possible, with the emergence of threads packages for user space concurrency.

2.10 Performance Limitations of OSI

The OSI model is designed specifically for *Open* systems, and so is non-optimal for homogeneous systems. The presentation level may be unnecessarily complex while the transport and network levels are designed to cope with public data networks where *access control* and *charging* may be important. They are also designed for old fashioned unreliable networks, and therefore contain a lot of protocol mechanism to deal with errors. There is no support for low-latency exchanges - a call setup is usually required at several levels.

A different layering model has been suggested by Zhang and Clark from MIT, to support mixed media traffic, and to provide low cost bulk data traffic as well as low latency exchange traffic. This is shown in Table 4.

5	presentation
4	buffering/multiplexing
3	rate control
2	sequencing
1	reliability

TABLE 4. Clark's Five Layer Model

The intention is to map more closely onto hardware and operating system characteristics. Svobodova also describes the importance of layers with respect to performance.[Svo84a] Almost all distributed systems like these mentioned are implemented over *connectionless* networks.

2.11 Traditional Communications Requirements

Traditionally, communication has three phases, while the form of service exchanges is modelled as a Request and Indication pair:

- Connection Establishment Phase + Connection Request and Indication

  This phase is concerned with providing synchronisation of the active end and the passive end, and with eliminating the chance of re-use of data packets from previous connections.

- Data Send and Receive Phase + Data Request and Indication + Interrupt Request and Indication

  This provides that data transfer actual service to the user.

- Disconnection Phase + Disconnect Request and Indication

  This phase can speed up the start of a subsequent connection by reducing the probability of outstanding data.

- Any Error Indications

Orthogonal to the service specification, there can be differing implementations of software for these interfaces:

- Procedural, possibly using upcalls

  Request and indication can be bound up in a single procedure call, with the call and return from a procedure providing each function. Any errors can be mapped into error values in a return variable. Upcalls **Mil88a** are a way of structuring a system symmetrically. An upper layer provides a lower layer with a procedure handle/closure in an initialisation phase, and the lower layer invokes this procedure for incoming events that it would normally provide on a polled basis. This is a very similar mechanism to vectored software interrupts.

- Structural

  Some systems provide interfaces with parameter lists. Others will provide an interface that forces the user to construct a structure or record often similar to the protocol data unit header - this can have efficiency gains.

- Message Passing

  Some systems explicitly implement the messages implied in the request/indication response model as separate messages between tasks/processes.

2.12 Service interfaces to Transport protocols

Procedural interfaces to the transport service are common; for example the (C language) system call interface to transport (and network) protocols in BSD 4.2 Unix **Lef83a** is:

```
#include <sys/types.h>
#include <sys/socket.h>

       s = socket(af, type, protocol)
        int s, af, type, protocol;

       bind(s, name, namelen)
       int s;
       struct sockaddr *name;
       int namelen;


       listen(s, backlog)
       int s, backlog;

       connect(s, name, namelen)
       int s;
       struct sockaddr *name;
       int namelen;

       ns = accept(s, addr, addrlen)
       int ns, s;
       struct sockaddr *addr;
       int *addrlen;

#include <sys/time.h>

       nfds = select(width, readfds, writefds, exceptfds, timeout)
       int width, *readfds, *writefds, *exceptfds;
       struct timeval *timeout;


       read/write/readvec/writevec
       sendto/recvfrom
```

Address structures should be adequately complex to support a range of protocol addresses. For instance the BSD 4.2 address structure:

```
struct sockaddr {
       u_short sa_family;              /* address family */
       char    sa_data[14];            /* up to 14 octets of direct address */
};
```

will *not* support full *ISO* addresses which can be up to 20 octets long, and also have a more hierarchical structure which is not reflected in the choice of a flat data structure.

The Service Interface may support non-contiguous buffers: Some systems, like Unix, support an **iovec** structure, which is a linked list of pointers to buffers. Such an arrangement is usually known as *Scatter/Gather* buffering.

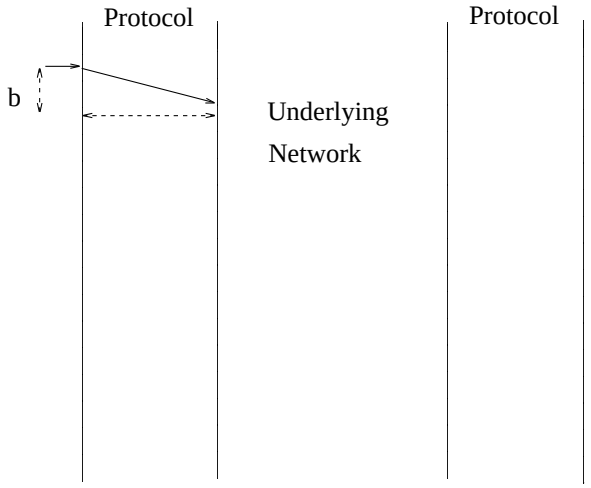
A service interface may be non-procedural. Commonly a single procedure will have a complex structure which contains all the information relevant to any service call, such as input buffers, output buffers, input and output buffer lengths, source and destination addresses, any address lengths needed and any flags (like expedited etc). Often this can be efficient, because this structure may map onto a connection descriptor within the underlying protocol.

2.13 Non blocking and Asynchronous interfaces

There are also various choices for the degree of concurrency between the user and the system implementing the service. The relative concurrency of user and protocol threads for Synchronous versus Asynchronous is compared in Figures 19 and 21. Similarly, the related behaviour of Blocking versus Non-Blocking interfaces is shown in Figure 20. Non-blocking interfaces are ones that allow the user's thread of control to continue before any *I/O* is necessarily complete. This allows the user to be part of the *pipeline* effect when the I/O is to a protocol.

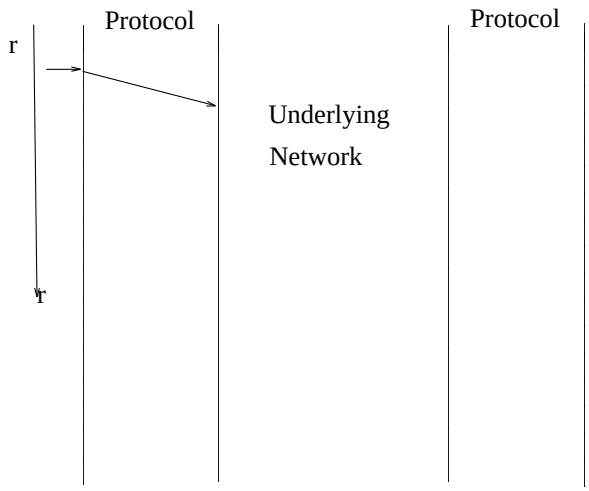
Asynchronous interfaces, on the other hand, allow the user to specify an interest in some I/O events, and to be *notified* when that I/O completes. Note that this is completely orthogonal to whether the underlying protocol is reliable or not. An example of the former would be *writes* to a stream returning before the data had reached the network, allowing the user to prepare the next buffer for sending. An example of use of the latter might be where a user wants to handle events from several sources, and might use asynchronous I/O to make the code simpler than polling.

BSD 4.2 Unix provides all these, and a polling interface using a general I/O system management call *ioctl* to set non-blocking I/O, the *signal* system (SIGIO) for asynchronous I/O and *select* to poll a channel for data. Select is commonly used to write simple schedulers based on up-calls.



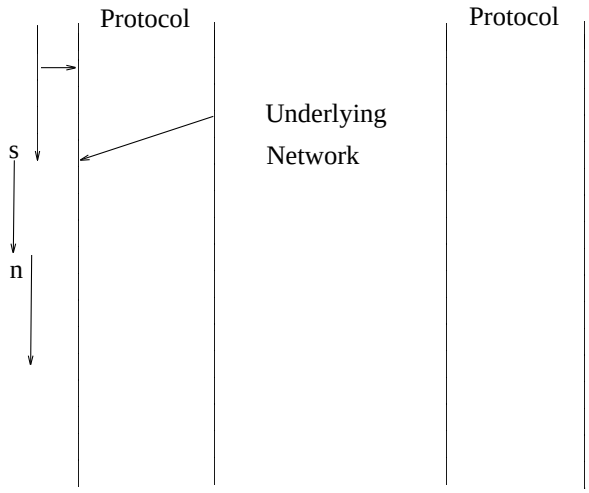
b=process blocked/suspended/not running/asleep

Figure 19. Synchronous I/O



r = process runnable all the time.
but still pre-emptable for other reasons.

Figure 20. Non-Blocking I/O



s = handle input event, usually change control thread
n = revert to previous thread

Figure 21. Asynchronous I/O

Different interfaces like these cause problems for buffer ownership - who can allocate, modify and free buffers when? Different concurrent programming languages and environments may make different assumptions about buffer ownership. This is discussed further in Appendix 6. It has been argued that asynchronous and non-blocking interfaces to Inter-Process Communication make the task of writing distributed programs much more difficult, and that blocking Remote Procedure Call systems are much more traceable.**Wil87a** Counter-arguments are beginning to appear, although this is still the prevailing view. **And82a**

2.14 Transport-to-network Level

This will reside within the operating system, since this level *must* have special knowledge of memory (within the application address space), and application process *liveness*.

2.15 Network-to-Link or Device Level

The network can provide a reliable service, for instance if it is a single X.25 network. On the other hand it may provide a less than reliable service - if it is a concatenation of X.25 networks via X.75 gateways, or if it offers connectionless network service. The ISO Transport protocol [ISO 8073] has a variety of *Classes* to support reliable connection oriented data transfer over this variety of network services. This design is worthy of closer examination.

2.16 ISO Transport Classes

In the ISO model, the transport layer is intended to mask all network dependent characteristics from the higher levels. A transport user specifies required **Quality of Service** in terms of Throughput, Delay and Signaled error rate (resets and packets lost)

The Transport layer is supposed to use the network layer in such a way as to provide this. It may use one network connection for several transport connections (multiplexing), or it may use several network connections for one transport connection (splitting).

In this model, there are three classes of network service, termed A, B and C in the standards:

A. Unacceptable error rate, no sequencing (i.e. connectionless)

B. Acceptable error rate (ie. very few undetected errors), unacceptable signaled error rate, sequenced. (e.g. connection oriented but using relays).

C. Acceptable undetected and signaled error rate, sequenced.

This leads to five classes of transport protocol to suit these network services (some also allow multiplexing and support extra functions such as expedited data). This is illustrated in Table 5.

NS Class	Non-multiplexing	Multiplexing
A		4
B	2	3
C	0	1

TABLE 5. ISO Transport Protocol Classes

ISO transport protocol features to implement these classes over the different network services, and provide a single service include:

- Class 0 (CCITT T.70)

  No expedited data, no recovery from network service failure and no multiplexing.

- Class 1 (Basic Error Recovery)

  Recovery from signaled network errors but no multiplexing.

- Class 2 (Basic Multiplexing)

  As Class 0 but with multiplexing and expedited data and flow control for each multiplexed channel and disconnect for each multiplexed channel.

- Class 3 (Error Recovery)

  As class 2 but with recovery from signaled network errors.

- Class 4 (Error detection and recovery)

  Includes detection of missing and mis-sequenced packets - sequencing Detection of errors not signaled by the network service by means of a checksum, expedited data, ARQ, selective retransmission and dynamic window management through a credit system.

Throughout this thesis, we argue that given the independent failure mode of end systems and the network, only Class 4 can provide a meaningful End-to-End Service. In our experience, Class 4 is also easier to implement than 3 or 2, since it does not have to manage connections (although when run above a CONS network, a connection management sub-layer is needed, of course).

2.17 Network Errors

As stated before, the network can introduce a variety of errors into packets and collections of packets: Noise or transmission errors or software can cause the bits in the packet to change. Congestion or intermediate error detection can cause packets to go missing. The network could duplicate packets or re-sequence them so that a receiver sees the packets out of order. Because of variation in packet delay, *unnecessary* re-transmissions from a transport layer may result in duplicates at the receiver. The network may break packets that the transport layer gives it into smaller parts, and only deliver some of those fragments. The network may take a long time to traverse, and yet have high possible throughput. In addition to all this, the network may simply break. To make the task of programming in the end-to-end layer more complex, the network may provide multiple connections between machines (e.g. X.25), or only single connections, or more simply, just datagrams.

The end-to-end argument leads one to suggest that while there are 5 classes of ISO transport protocol, perhaps only class 4 is actually functional as a true end-to-end Protocol. The less functional classes depend on **signaled** errors from the network. Yet by definition in a distributed system, components have independent failure mode. Thus the network and either end system may fail **without** signal. This means that only the end system can detect this, and only by a mechanism such as timeout, which maps from *lack of signal* to signal.[12]

2.18 What must a Transport Protocol do?

A transport protocol must have *procedures* and control fields in packet headers to deal with all these problems. It should, on the other hand, be reasonably efficient. A Checksum on the whole transport protocol data unit (TPDU) carried by every packet **detects** errors, allowing the receiver to discard *most* corrupted packets. **Sequence numbers** and **timeouts** carried by every packet allow the receiver to detect missing packets. If packet 3 arrives after packet 1, the receiver knows packet 2 is missing. Two mechanisms exist for the transmitter to realise that it should retransmit packet 2: Positive Acknowledgement and Negative Acknowledgement.

Positive Acknowledgement relies on the transmitter retransmitting packets after a *timeout*. This is used in ISO TP. Negative acknowledgements allow the receiver to inform the transmitter that a packet is missing, and require the receiver to have a timeout which triggers a NACK. Sequence Numbers allow a receiver to determine when packets arrive out of order and to detect duplicates. Sequence numbers appear in all packets where we care about ordering (e.g. connection setup, tear down, data and acknowledgement), but need not appear in other kinds of packet (e.g. resets and aborts). Sequence numbers are usually fixed size in some number of bits, and therefore can only re-order within some range.

The Transport protocol must be able to break user requests to write or read data into smaller chunks for the network level.

---

12. An incident on the JANET 2Mbps X.25 network in mid 1991 is salutary: A new release of X.25 software caused the network to deliver data packets out of sequence. TCP/IP hosts continued to function with slightly reduced performance (as underlying X.25 circuits were broken and re-made), while no ISO/X.25 host could maintain a connection for a significant period of time. The problem would not have existed for an TP Class 4 on X.25 host.

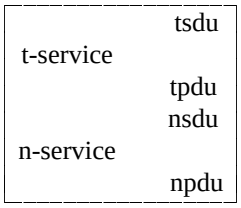


TABLE 6. Packet Size changes through Layering

Table 6 shows the concept of service and protocol entity. In the ISO model, between a one protocol layer and another, we find two reasons that packets may have to be copied and new headers constructed. One is to do with the different control information available to ascertain what the appropriate packet size for a given protocol entity to use. This is called *segmentation* and *re-assembly* in ISO transport protocol. The other is that a service interface may permit arbitrary size service data units, when the protocol has to constrain PDU sizes to those available from the lower layers (eventually determined for good by the technology - some hop on a path will have a minimum packet size). This can obscure this choice and lead to serious inefficiencies, and this is discussed further in Mogul's work**Moga**

Windows are used to use network bandwidth more effectively by allowing some number of packets to be outstanding - i.e. not acknowledged. This is also known as *pipelining*. In addition to this, the application will require flow control between each end. This is done in the service interface and by setting window sizes appropriately.

Busy Idle traffic can be used to detect the networks *liveness*. This is often done by retransmitting the last data packet, or acknowledgement. Multiplexing is done by ISO TP class 4 by reference numbers. Source and Destination Reference numbers identify a particular connection between two machines. To detect old (*stale*) connections, freezing of reference numbers from half broken connections is used.

2.19 What else do you need in a Protocol?

A connection oriented transport protocol keeps all this information in a *connection descriptor block* and needs one per connection if multiplexing is happening. The connection descriptor in general will hold the Source and Destination N-SAP and T-SAPs + Reference Numbers and transmit and receive Sequence Numbers and Windows. It is used to access queues of transmitted packets waiting to be acknowledged, as well as queues of received packets waiting to be handed to the user in order. It also holds timer values (e.g. retransmit timer value, acknowledgement delay timer) and the basic current State of connection (open/closed/data), and anything else necessary (for instance connection statistics! In chapter 3 4 and 5, we examine this state and how clearly identifying it can simplify implementation and protocol performance analysis.

2.20 Related Work

Two areas of work elsewhere are particularly relevant to that described later in this thesis. They link important components of a protocol in terms of implementation to the design of the communications infrastructure of a kernel. These are outlined below.

2.20.1 The x-kernel

The x-kernel **Hut88a** is highly tuned to protocol implementation, providing specially optimised functions for:

- A message manager

  This is essentially a heapstore system with special functions for joining/splitting/prepending to provide rapid header construction and packet assembling/disassembling.

- A message map manager

  This provides a mapping from ID to process that enables multiplexing/de-multiplexing through layers to be carried out very efficiently (implemented through a tuned hash table).

2.20.2 The V-kernel

The V Kernel [Che85a] is a higher level system, operating at the level of page level access between files and processes through the provision of the single protocol system, VMTP. V builds an RPC system on top of VMTP and most usefully, provides process group communications using group addressing and underlying IP multicast to carry packets to more than one machine.

One level further up, Cohrs et al[Mil88a] describe a way of structuring the interface between the transport protocol and applications using *upcalls* which are analogous to soft interrupts (Unix signals). As with the X Windows System, the programmer provides pre-installed handlers for events of interest, and the protocol interface uses these when demultiplexing incoming messages.

2.21 Performance Related Work

Work on analysing a given protocol for the performance critical paths is exemplified in Jacobson.[Jac90a] Jacobson's main aim is to compress the TCP and IP packet headers, so that there is less overhead transmitting them on low bandwidth serial lines. In trying to choose which parts of the headers could be compressed, an information theoretic analysis similar to that that leads to Huffman coding was carried out. This led to finding the most common case paths through the send and receive code of the BSD TCP implementation, and the careful optimisation of those paths. This analysis is known as ˜header predicition", and is based on the observation that certain sequences of packets are much more likely in normal operation than others. The approach is quite generally applicable.

Feldmeier and McAuley[Fel90a] report similar findings, in a more general piece of work on protocol ordering. They examine how the *order* in which one layer needs to process things has an important consequence for the *performance* of the next layer up or down. For instance, they show that a layer that is doing cipher encryption needs entire blocks (in the size it dictates) before it can proceed. This necessarily leads to longer latency times.

Wakeman et al report on practical experience, finding RPC performance problems directly caused by excess hiding of information through layering, re-enforcing this viewpoint.[Waklya]

3. Chapter 3: A Sequential Exchange Protocol

In this chapter we present the design of the Sequential Exchange Protocol, which we call ESP[13], a Transaction Protocol for RPC. We look at the need for Presentation level Integration, the need to support recursive calls, and then look beyond RPC (Messaging/Streaming).

This protocol could be seen as a replacement for TCP, VMTP[Che86a] or XTP[Inc90a] or a suitable improvement on the pseudo transport protocol used on UDP for support of NFS/Sun RPC. This latter approach currently uses manually configured timers and packet sizes. The use of TCP would automate this. However, TCP was regarded as too costly for the short exchanges typical of small file access. Recent TCP enhancements mean that this is not clearly the case anymore, and techniques such as caching of connection descriptors mean that very fast transaction systems are now built using TCP. However, highly distributed programs may have very dynamic patterns of client-server interaction, which defeat such caching, and still require a protocol with properties somewhere between UDP and TCP.[Inc86a, Mic89a] Further useful background and general justification for specialist protocol support for RPC systems may be found in Braden.[Bra85a]

ESP was designed to provide optimal transport protocol support (as described in this chapter) for the RPC system developed for the Alvey Admiral Project, in accordance with principle of frugality in software engineering. [Bac86a] Since its design (at around the same time as VMTP, and somewhat before XTP), there has been a little progress for transaction and RPC support in the Standards or Internet communities. Work on a Class 5 of ISO Transport Protocol has recently begun. ESP itself is no longer in use due to lack of support. The ideas present in the design can be seen in the TP5 work.

3.1 Introduction

This chapter describes the specification of the Sequential Exchange Protocol. [14][Bac8.a] This end-to-end protocol is designed to support sequential exchanges of paired messages, which may be large. The protocol is designed with a view to balancing the requirements of reliability and performance over local area networks and over inter-networks with wide variance of delay and throughput.

The pattern of communication in a distributed programming environment is different from traditional applications. It can vary between single short accesses with fast services (e.g. time servers), many fast accesses to medium speed services (e.g. file servers), and occasional long accesses to slow services (e.g. graphics servers). All these accesses are characterised by a sequence of exchanges of data, where requests are completely delivered, some processing is done and a reply is completely delivered.

Many typical distributed systems use a Remote Procedure Call system to support these accesses. Some distributed systems also provide one way message passing, with no reply from the application. This is a subset of the procedural case.

Many implementations of RPC mechanisms use unreliable datagrams for local networks and reliable stream protocols to deal with unreliable networks and to handle large parameter blocks. The problem with this approach is that RPC should be a tempting tool for the distributed programmer. No knowledge of the underlying communications medium should be required, so this choice should either be automated, or dispensed with. A transport protocol that unifies RPC requirements with network adaptability would obviate this knowledge and choice.

---

13. *"The fundamental unity of the Sequency and Simultaneity points of view became plain; the concept of interval served to connect the static and dynamic"* - Ursula LeGuin in The Dispossessed.

14. This protocol was specified by the author. Three implementations were carried out, one by Mark Riddoch under Unix, and two by the author under the CMOS [Cro86a] and PC/IP/MS-DOS operating systems. Much credit is due to Mark Riddoch and Ben Bacarisse for assistance in debugging these implementations.

To this end, ESP was designed.

3.2 The ESP Protocol

ESP is designed, and has been implemented, as a sequential exchange protocol that supports the transfer of large amounts of data, without the full state required for connection protocols. [For instance ESP does not have or need three way handshakes for connection making and breaking, or call collision handling. ESP does not support concurrent two way traffic.]

The protocol is designed to use a connectionless network layer such as DARPA Internet Datagrams, or the ISO connectionless network service. This stack is illustrated in Figure 22. ESP stands (but is not an acronym) for the Sequential Exchange Protocol.

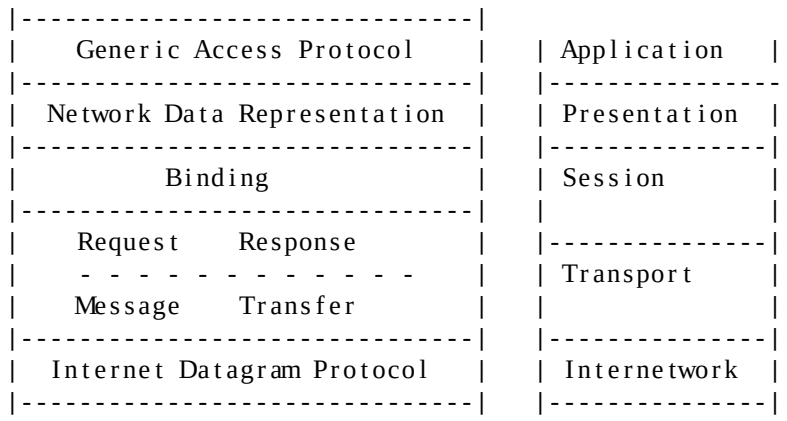


Figure 22. ESP - Protocol Stack

The following criteria underlie its design. The order is significant in the importance of a design choice.

- In keeping with the principle that no protocol layer should duplicate work done by a higher level, ESP assumes that servers that wish to maintain state information for clients will do so. ESP does not do so for them, any more than the minimal state to provide adequate reliabe transer of data.
- The protocol detects and filters duplicate messages.
- The protocol informs new or existing clients of absence of servers where possible, and does not burden servers with the job of replying reliably.
- For small parameter blocks and local communication, the protocol ideally exchanges only two packets, one for the call, one for the return. (Reduced Packet Overhead)
- The protocol can exchange very large request/reply messages.
- The protocol works over an internet with a high variance of packet size, delay and throughput.
- The protocol makes the presentation layer (RPC "stub code") easy to build and efficient.
- The protocol has a one basic packet format to simplify implementation. (Reduced Packet Type Set)

The natural structuring of the protocol specification is in two sublayers. The first deals with reliable end-to-end delivery of the data. The second deals with request/reply message association. Apart from simplicity, another reason for the separation of the message and exchange layers in the design is that a different sublayer may be put above the message layer in the future to support other Inter-Process Communication (IPC) semantics.

Section 3.2 describes the end-to-end delivery layer. Section 3.3 describes message association and client/server crash detection. Section 3.4 describes the kind of service interface that ESP provides. Section 3.5 gives some simple examples of the request/response system working. Section 3.6 describes an

implementation’s performance.

Finally the packet formats are given, together with a state table for the end-to-end delivery sub-layer.

3.3 The Message Transport Sublayer

There is a single ESP implementation for each host on a network, which multiplexes and demultiplexes messages between clients and servers on different hosts. Ports are the basis for client/server addressing. Features of the message sublayer include:

- Addressing

  Messages are addressed to a port on a host. The message header includes source port and destination port on a host. There are (2**16) ports.

- Ports

  Ports for receiving messages on, and transmitting messages from are normally allocated by the protocol. Ports 0 to 1023 are reserved for well known services. The most common use for a well known port is the address of the ubiquitous name/location service, with which most services using ESP are registered by name. [It is up to the distributed system programmer to provide an appropriate name service].

  Ports are allocated by ESP from 1024 upwards and wrap round. New users of ESP get new ports. Between machine crashes, there is no guarantee that these allocated ports (i.e. not-well-known) will be unique.

- Sequencing

  To filter duplicate messages, sequence numbers are used. Each message sent from a given port uses the next sequence number. Messages in sequence from a given port bear consecutive sequence numbers modulo 2 ** 32.

  If a client process crashes having sent a message from a well known port, or a machine crashes having sent a message from an ESP allocated port, a different scheme is used to insure duplicate filtering. After 2 ** 32 messages, messages would normally be dropped as duplicates. The request/response sublayer can deals with these problems.

- Large Messages

  ESP supports message sizes of up to (2 ** 64) + (2 ** 16) octets of user data. To achieve this over a network that supports small packet sizes, ESP breaks the message down into segments which contain the segment length and octet offset position of the segment in the whole message. The offset can be up to (2 ** 64) and the segment length is up to (2 **16) of user data. The actual maximum segment length used depends on the network service. In the case of DARPA IP this is 576 octets less IP and ESP header sizes.

  The choice of 2**64 was to match the address space of emerging computer architectures such as Cray and DEC Alpha. An inter-process communication protocol should support this fully.

- Retransmission and Acknowledgement

  The ESP packet header contains an acknowledgement field which acknowledges up to the current octet received. [NB This really limits the maximum message size to (2 ** 64) octets]. The sender sets a timer for each segment sent and retransmits any segments not acknowledged when the timer goes off. ESP never acks parts of segments.

  If a segment has to be retransmitted too many times, the whole message fails and ESP reports this to the user. The maximum retransmission count for the first segment is larger than for following ones.

  The receiver sets a timer (which is smaller than the retransmission timer) to delay acknowledgement in case further segments arrive and can be acknowledged in a single packet.

This timer is larger for the last window of segments for a reason that is explained in the next section. The acknowledgement of the entire message is implicit in the acknowledgement of the last segment.

- Windows and flow control

  ESP uses an adaptive window scheme for end-to-end flow control. Since there is no open exchange, a first guess transmit window size is used. Every ESP packet carries its sender's current receive window. The receiver may adjust this guess in any packet it sends to the transmitter. The window can vary between 0 and 2 ** 16 packets. The receiver also stores the window size advertised by the transmitter for use in reply messages.

  ESP has a low and a high water mark of receive buffers for each port. Whenever the high watermark is exceeded, the window is decreased proportionately. Below the low watermark, ESP will increase the window size to the original, or larger if information about the network route is available.

  If an out-of-sequence segment is received, ESP will decrease its window, and flush any acknowledgements for that destination. Packets outside the window are not rejected or dropped unless there is no buffering for them, but they will affect a new window advertised.

- Error Packets

  These are discussed in the next section.

3.4 Pairing Messages for Request/Response

This sublayer has several jobs. Firstly, request and reply messages are associated. Then ESP uses subsequent messages to acknowledge the last segments of previous messages. The detection and handling of messages from a port using stale sequence numbers is handled here. Busy/Idle traffic - pinging - is run to detect server status between request and reply. Finally errors of various kinds are propagated to the layer above to help with exceptions.

The message level only keeps state while sending or receiving a message which consists of the send and receive window information for each source/destination pair.

The exchange level only keeps state during a transaction. Only the server end of a transaction needs this state, since the message level provides reliable enough delivery for the client. Elements that go to make up the request/response sublayer include:

- Association.

  Two special message types are defined: Request and Response. A response message may only be sent from a given port by the ESP user if the last message it received on that port was a request. An ESP user may receive requests while listening for replies, so that call/replies may be nested.

- Conversation id.

  Every message carries a *conversation id*. This is seeded from time at boot time, and is between 0 and (2 **64) - 1. The current implementation reboots with the high 16 bits of the *conversation id* set to the current time in milliseconds. A new value is put in the message for a new call message. Replies carry the same value. If a call is received on a port while a reply is also expected, replies to that incoming call carry the same ('stacked') *conversation id*.

- No Open

  The message layer assumes no relationship between messages. The exchange layer assumes no relationship between exchanges. Nevertheless, ESP is not a simple connectionless protocol. An exchange of large messages resembles alternate simplex use of a connection protocol. For instance, there is constant negotiation of window sizes during message transfer.

On many systems, a *listen* on a port involves a lot of resources. Because of this, the *listen* for a reply should also be able to handle incoming calls. Typically, a remote procedure call may result in calls of further procedures, which could easily lie in the same process as the original caller.

- Call Back

  If the message header includes a *conversation id*, not only can this allow calls to arrive on the reply port whilst the application is listening for a reply, but it can also allow these call-backs to acknowledge the original request message.

  The only restriction on call-backs in this system is that they only originate from a process further down the tree of calls. The rules for nesting calls and replies exactly map the way procedure calls can be nested and recursive.

- Implicit Acknowledgements

  ESP dallies before making acknowledgement for longer on the last segments sent (as stated in the previous section). This is in anticipation of being able to send a quick reply from a service if the message was a request, or having a new call for the same service, if the message received was a reply.

  In many cases this will achieve a two packet exchange for small message calls over a local network.

- Resequencing

  Resequencing is used to deal with rebooted/restarted clients sending from old (stale or well-known) ports, with outdated sequence numbers.

  Old sequence numbers are detected by the fact that the *conversation id* is same.

  When an old call message is detected, ESP sends an error packet with a suggested new sequence number to use to the offending source. This is larger than the expected sequence number to allow for network transit times. It is up to the sending ESP whether it accepts this, but messages will not be accepted until it does.

  This limits the maximum rate at which clients using well known source ports can restart, and a maximum rate at which crashed hosts possessing such clients can reboot.

  Since calls and replies carry the same *conversation id*, replies that have persisted in the net from a previous caller will not cause resequencing.

- Pinging

  Pinging is used to detect server crashes. It is used between ESP peers as a control mechanism, and is not part of the service interface. In some exchanges there may be a long delay between the call and the reply while a server is processing. The network, the server host or the server process may be unreliable. Pinging is required to detect any of these failures.

  Ping packets and acknowledgements only start up after a message has been fully and explicitly acknowledged. Ping packets also carry the latest window sizes. The request response should only fail if more than a certain number of pings in a row are not acknowledged.

- Error Messages

  ESP has 8 different errors that can be reported by a receiver to a transmitter.

  - No error

    This is used merely as a mechanism for being able to transmit no-op packets, for purposes such as changing the window size.

  - Not Implemented

An ESP Packet type has been received that is not recognised.

- Resequence Message

  As described above. The optional data field includes a new sequence number.

- No Listen

  There is no *listen* on the port the message was sent to.

- Receiver Busy

  There is a *listen*, but the application is between receiving a call and sending a reply, which may be in a different conversation.

- Unexpected Reply

  A reply message was received when no request was outstanding. This should not happen if the sequencing and *conversation ids* are functioning properly.

- Server Down

  The Server has crashed sometime between receiving the first part of a call and sending a reply.

- No Resources

  The receiver has run out of some resource or other.

All errors except the no-op and optionally the resequencing message result in the exchange aborting and the layer above at each end terminating in default circumstances. (Using complex recovery is a higher level decision).

The total state ESP holds for each client waiting for a reply is a stack of the source address of the responders, their current *conversation id*, with ping timers and counts. As replies arrive, this state is unstacked. The total state held by ESP for each server, while the application is processing, is a single address to reply to. In addition, a server may switch roles to being a client (of anyone) between receiving a request and sending a reply, as befits a system that permits RPC callbacks.

3.5 Service Interface

There are three aspects to the service interface that influence the way the protocol works. These are buffering, the user exerting flow control and the user aborting exchanges.

- Buffering

  The protocol maintains implicit marks where users have pushed parts of messages, so that data is received in the same units as it was sent. This is so that presentation level atomic types are never divided.

  Ideally ESP hands buffers to the layer above on input, and vice versa for output. This is not feasible in some operating systems. No assumption is made that the transmitter has all the message ready to send. It may be sent and received piecemeal. The parameters to an RPC are not known until call time, so a receiver cannot know how much buffering to have ready. Mapping of machine types to net types and vice versa takes an unknown amount of space until some data is received that gives the receiver more information about buffering. ESP can support presentation levels with explicit and implicit typing/sizes handed by presentation/stub level.

- Flow Control

  The receiving user may exert flow control implicitly by providing less receive buffering than the message requires. The service interface allows the user to tell the sender to proceed by increasing the window.

- Exceptions

The client server model is very asymmetric. ESP supports this asymmetry by realising the fact that servers should not be concerned with replies getting back to clients. ESP tries very hard to provide requests and replies to the application once or not at all. However, since ESP operates over unreliable networks, it may (despite all efforts) be unreasonable to provide a satisfactory service in terms of performance.

The service interface allows the client to abort during the *listen* for a reply. ESP then does its best to propagate this to the remote end (as a "no listen" error) to clear any state there quickly.

3.6 Examples of Operation of Request/Reply

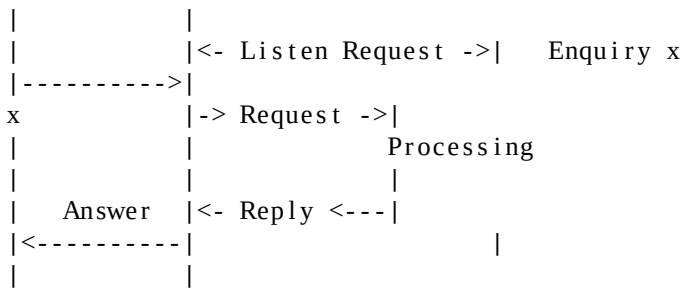


Figure 23. Simple Exchange

Figure 23 illustrates a minimal packet exchange which consists of a single request and a single response packet. The reason this works can be seen in the context of a sequence of transactions, rather than a single isolated exchange, as shown in Figure 24:

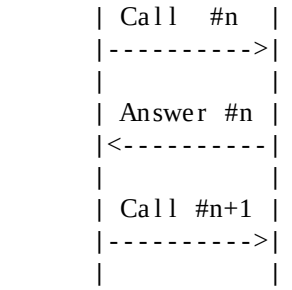


Figure 24. Sequence of Exchanges

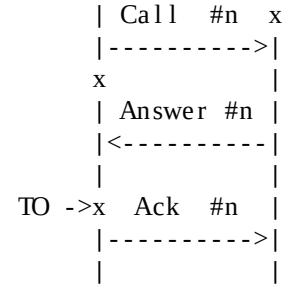


Figure 25. Exchange with Answer Acknowledgement

Figure 25 shows what happens when there is an hiatus in the sequence of requests/calls. The sender times out and acknowledges the answer packet.

Clearly the symmetrical situation is TO in 'server', leading to Call Ack. The fourth case is both client and server processing a lot between calls and answers; then both TO's operate, and there is a 4 way exchange. Figure 26 illustrates this maximum packet exchange (in the absence of packet loss, of course).

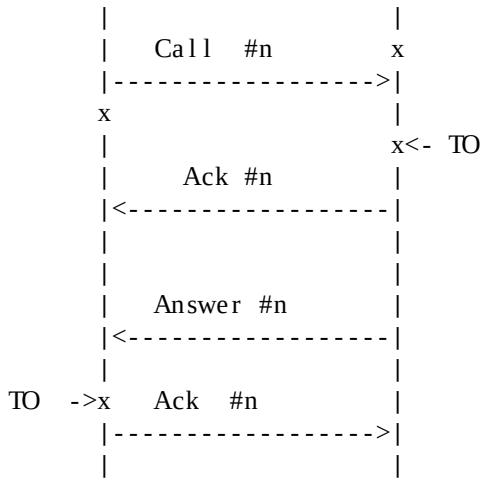


Figure 26. Exchange with Call and Answer Acknowledgement

The picture is further complicated by retransmissions. Clearly they must only be used to elicit the first ack, nack or reply from the server when calling, and ditto when replying.

Now we add segmentation and windows. The only trick here is to handle the transition from send to receive state correctly in the "client". Replies as implicit acks can only be accepted when all the request has been sent.

After that, the roles of receiver and sender swap.

Figures 27 and 28 show the (loss free) packet exchanges when the request and responses at the higher level (e.g. RPC parameters) exceed the packet size for single packet exchanges. Figure 27 shows the case for explicit acknowledgements (no recent or closely subsequent call, and relatively long processing in the server). Figure 28 shows the case for implicit acknowledgements (stream of successive calls and short processing time in the server).

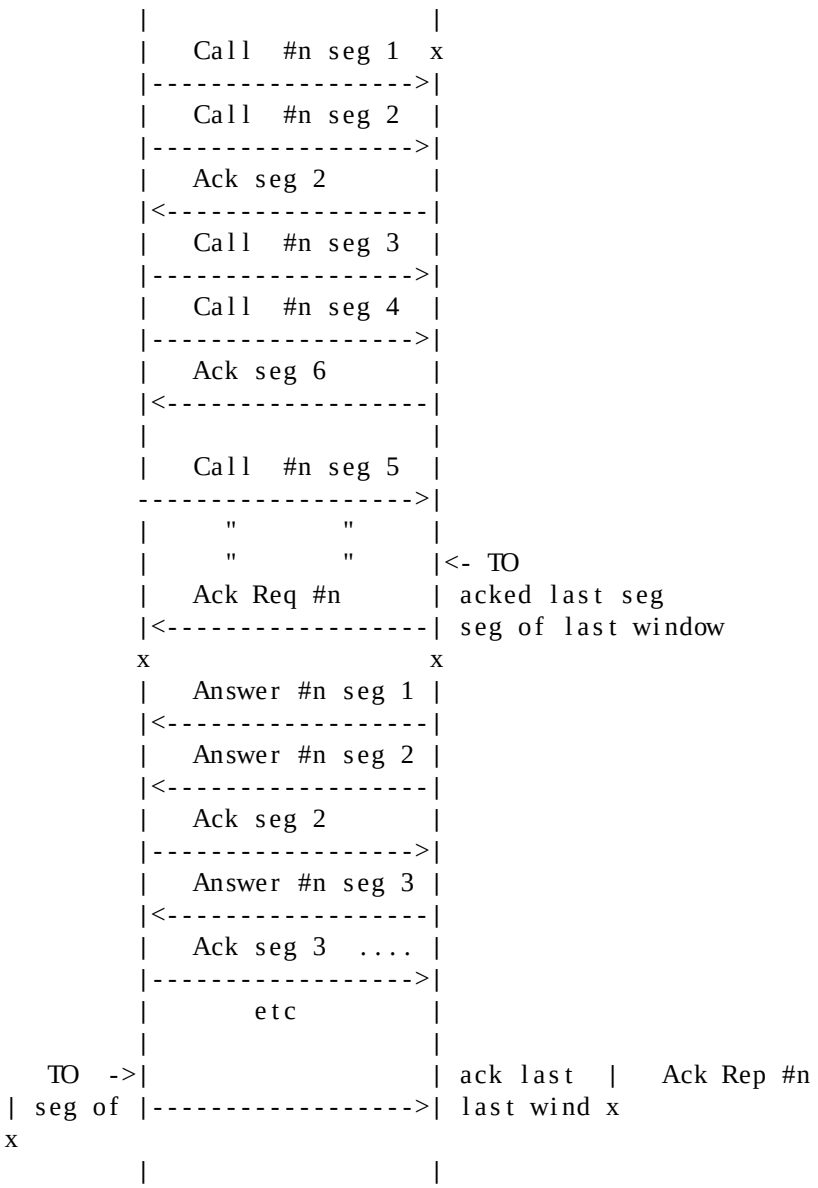


Figure 27. Large Request and Response with Explicit Acks

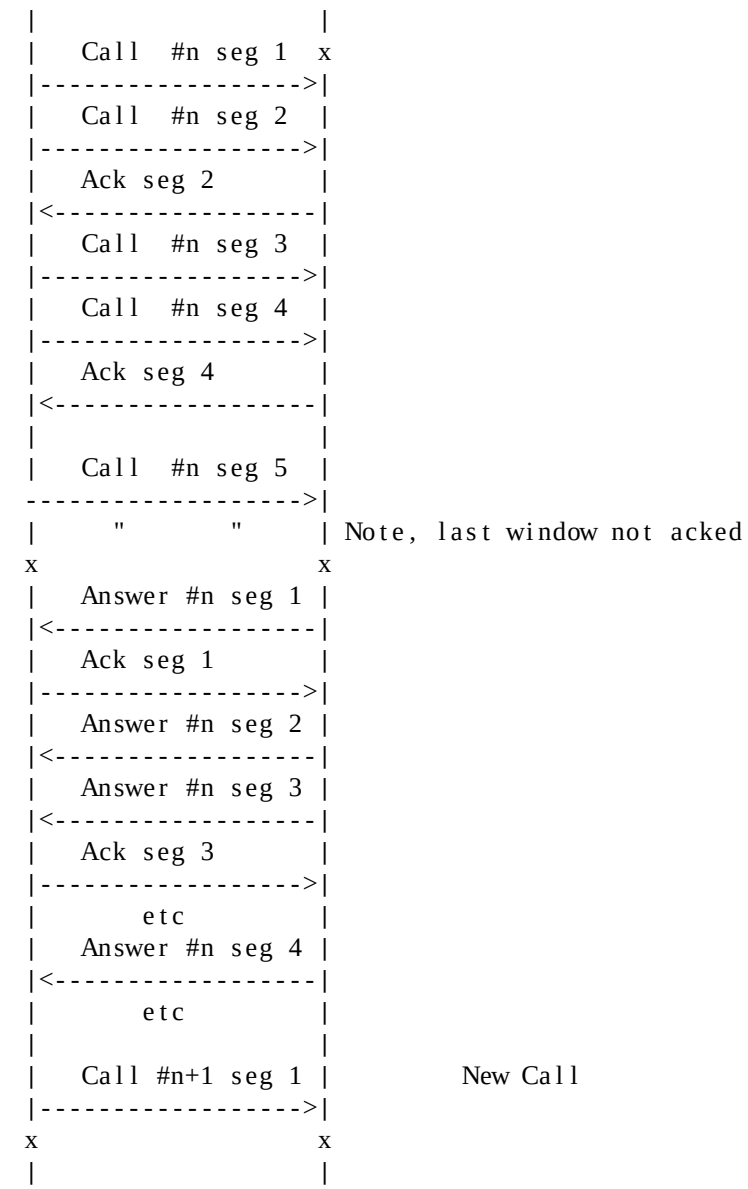


Figure 28. Large Request and Response with Implicit Acks

3.7 Performance and Possible Enhancements

The first implementation was located in the kernel (as a socket task) on a Sun 2 running 4.2 BSD Unix. It had a process to remote process round trip delay of 15-20 milliseconds for the ideal two packet exchange over a 10 Mbps Ethernet. For large messages the throughput was around 500 Kbps of user data.[15]

The following are experimental enhancements:

- Selective Acks

  For efficient use over the Internet, to avoid the systematic overrun problem when going through bridges and gateways, ESP will introduce a selective segment acknowledgements and retransmission system.

  Acknowledgements include lists of packets received out of sequence up to reasonable limits. Since no acknowledgement except the one for the last segment contain data, this is easily implemented by appending the list to the acknowledgement header and flagging the header.

- Type Of Service

  Three message semantics are defined: *idempotent*, *non-idempotent* and *non-blocking*. The application chooses one of these, which map onto no-duplicate- filtering versus duplicate-free behaviour respectively in the receiver (i.e. the message numbers are/not ignored), versus allowing multiple outstanding calls in clients.

3.8 Forwarding Calls and Multicast

We now discuss call forwarding, and the possible support of multicast calls.

ESP support for forwarded calls is limited to the *conversation id* support for servers and clients to swap roles, and call-back. The VMTP system of forwarding calls to other servers without replying, and expecting the second server to reply is not supported.

To function as a reliable multicast over IP multicast depends whether all or one of reply messages are required by an application. For replicated database systems, ideally ESP would support the collection of all replies, and multi-destination retransmission/flow control. This is the hardest case, and the simpler cases will fallout in any solution to this.

This would need at least the same amount of state as for doing multiple unicasts, and so has not been designed in. In any case the application that needs to collect multiple replies is likely to maintain a great deal of this kind of state anyway. This is dealt with in more detail in the next chapter.

3.9 Packet Layout

The Sequenced Exchange Protocol, has three variants of the basic packet format which are:

- Data packets: Calls, replies and acknowledgements are sent in these packets. The packet format and flags value assignments are shown in Figures 29 and 30.
- Error reports: These packets contain error information which has to be transmitted back to the other party in the conversation. This is illustrated in Figure 31. The Error Reason Codes are tabled in Figure 32.
- Ping packets: These are packets which are sent between the hosts involved in the conversation in order to ascertain whether the remote party is still alive. This is shown in Figure 33.

---

15. This compares favourably with current RPC performance over TCP in the OSF DCE, which is reported as similar, on machines with over 10 times the CPU, Bus and Memory performance.

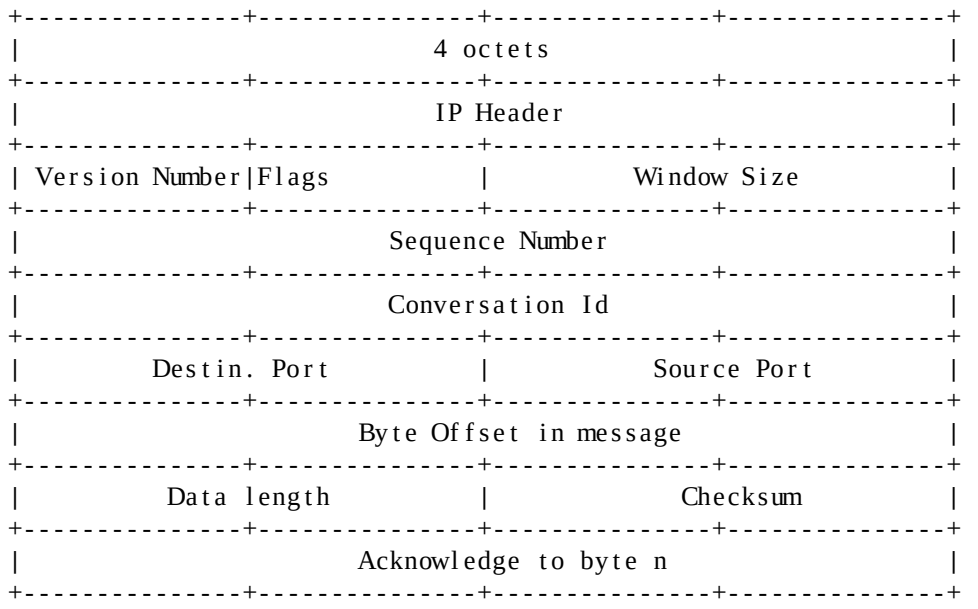


Figure 29. Data and Acknowledgement Packet

Bit	Name	Meaning
0	SEP_REQ	This is a request packet
1	SEP_LAST	This is the last packet of a message
2	SEP_ACK	This packet contains an acknowledgement
3		Unused
4	SEP_SEL	A list of selective acknowledgements follows
5	SEP_DATA	This is a data packet
6	SEP_ERROR	This is an error report
7	SEP_PING	This packet is a ping packet

Figure 30. ESP Flag Bits

- Version Number

  This version of ESP, currently 1. An 8 bit field.

- Flags Field

  This 8 bit field indicates the type of packet this is, as coded in the table above.

- Window size

  The current receive window size in octets. A 16 bit field.

- Sequence Number

  The 32 bit sequence number of the current message, carried in every packet header for this message.

- Conversation Identifier

  A 64 bit indication which conversation this message is part of, and an aid to sequencing.

- Source and Destination Ports

  16 bit fields identical to UDP ports.

- Octet offset in message

  A 32 bit field indicating where this segment fits in the overall message

- Data Length

  The length in octets of the data portion of this segment.

- Checksum

  If zero, not implemented, otherwise calculated in the same way as a UDP checksum. The complement of a 16 bit ones complement sum of the ESP header, data and IP overlayed header.

- Acknowledgement

  Up to which octet of data have been received, when this packet is an acknowledgement.

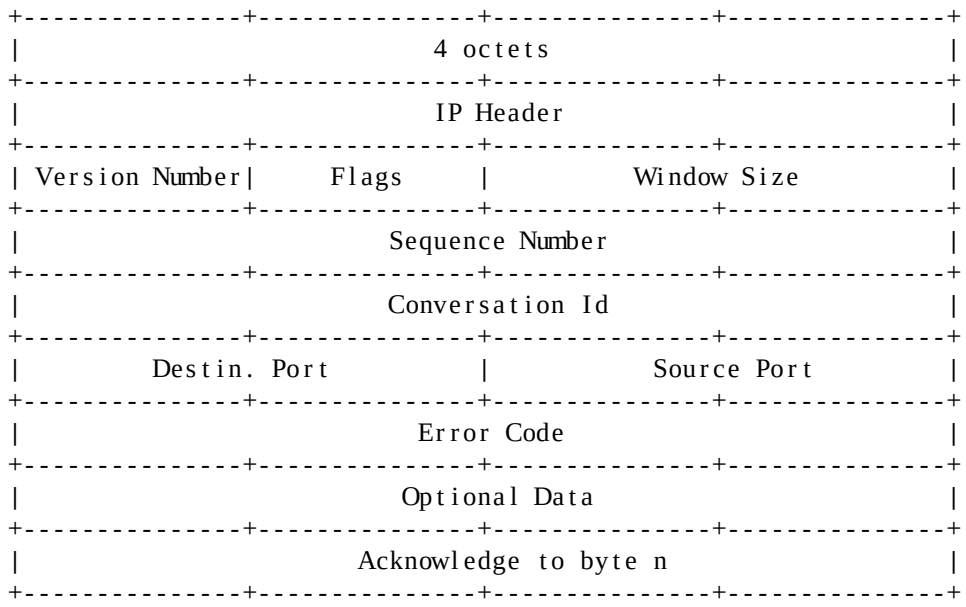


Figure 31. ESP Error Packet

Value	Name	Meaning
0	SEP_NOERROR	No error has occurred
1	SEP_ENOTIMP	Request you made not implemented
2	SEP_ESEQ	Please resequence with me The optional data field contains the sequence number that should be used.
3	SEP_ENOTLISTEN	There is no listener on that port
4	SEP_EBUSY	That port is busy
5	SEP_UNEXPECT	Your response was unexpected
6	SEP_SERVERDOWN	The server has gone down

Figure 32. ESP Error report codes

All fields as for the Data/Ack packet, except:

- Flags Field

  The 8 bit flags field is treated as an integer with values with allocated meanings in the table above

- Optional Data Field

  A 32 bit field for indicating a new choice of sequence number suggested by the error packet sender. No other value is defined for this field, and it should only be used when the error flags value is SEP_ESEQ.

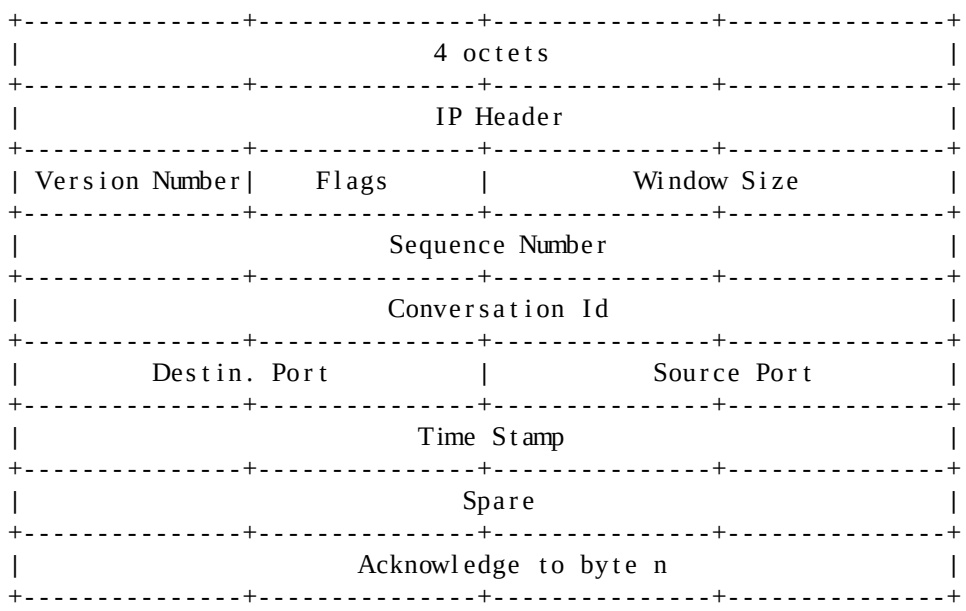


Figure 33. ESP Ping Packet

The only new field here is the timestamp field, which is filled in by the transmitter, and can be used by ESP implementors who have an accurate network timeservice for round trip delay calculations.

3.10 The ESP message sub-layer state table

ESP is designed around a Finite State Machine to enable ease of implementation and testing, and to permit future work on checking that all states are reachable and that the protocol is free from deadlock. This is illustrated in Figure 34.

	Rx State			Tx State		
	Idle	Rx	Error	Idle	Tx	Error
Rx 1st	If New Ack AcksLst GiveUser -> Rx Else ReXmit	If Interloper Notify Busy Else Drop -> Rx	Ack GiveUser -> Rx	NOP	NOP	NOP
Rx ~1st ~LST	NOP	If Interloper Notify Busy Else Ack If NotSeen GiveUser	NOP	NOP	NOP	NOP
Rx Lst	NOP	If Interloper Notify Busy Else If SeenAll GiveUser SetAckTimer -> IDLE Else Drop	NOP	NOP	NOP	NOP
Rx 1st AND Lst	If new GiveUser SetAckTime Else Drop	Notify Busy	If new GiveUser SetAckTime Else Drop	NOP	NOP	NOP
Tx ~LST	NOP	NOP	NOP	Set Retry Timer Tx Pkt -> Tx	Set Retry Timer Tx Pkt	If First Set Retry Tx Pkt -> Tx
Tx LST	NOP	NOP	NOP	Set TxAck Timer Tx Pkt	Set TxAck Timer Tx Pkt -> IDLE	-> IDLE
Rx Err	NOP	If Mine -> ERROR	NOP	NOP	If Mine -> ERROR	NOP

Figure 34. ESP Message Sub-layer State Table

Protocol Timer Events include:

- Retry Timer - timer with short period, after which packets will be retransmitted if not acknowledged.
- TxAck Timer - timer with longer period, after which time the last packet in a message will be retransmitted if not implicitly or explicitly acknowledged.
- SetAckTimer - timer with a period just less than the TxAck timer, if an implicit ack., ie a new message, has not been set when this timer goes off then an explicit acknowledgement of the entire message will be sent.

3.11 Default timer and counter values

First Segment Retransmission Counter	32
Retransmission Timeout	1 sec
Subsequent Retransmission Counters	16
Last Retransmission Timeout	4 secs
First Guess Window Size	8 * segment size
Ping Counter	10
Ping Timer	2 secs

TABLE 7. ESP Default Timer Settings

Table 7 lists the initial default timers in the absence of information from the local network interface configuration, or from routing table information. They adapt during the course of exchanges, and results may be cached in the routing table for future use.

3.12 A Note on Bit Ordering

Bit ordering in fields in ESP headers on the network is defined in exactly the same way as for the internet datagram protocol. (See RFC 791 Appendix B for details.)**[Pos81a]**

3.13 Crash and Reboot Protocol Interaction

ESP uses ports, sequence numbers and *conversation ids* to detect any duplicate messages (from retransmissions and network latency).

Clients and servers may crash between calls and replies. This may be due to machine crashes or process crashes. Clients and servers may also be using well known ports, or may be started in a particular order at boot time, also giving a systematic relationship between ports and processes.

This gives eight cases to consider:

1. Client process crashes and restarts:

   ESP will start this client's messages with sequence number 0. However ESP will in all circumstances give the restarted client a new *conversation id*, so all messages from this client will be new.

2. Client process using well-known port crashes and restarts:

   Again, although the client source port and sequence number may appear to servers as old ones, the *conversation id* is new, so no confusion with old messages ensues.

3. Client host crashes and reboots:

   Clients messages all start with sequence number 0. But all *conversation ids* are new.

4. Clients using well known ports when host crashes and reboots:

As above.

5. Server process crashes and restarts:

   In general the server will be allocated a new port, which will be found by clients using some nameservice. All old clients should be notified of the death of the old server by ESP. All clients of the restarted server are new.

6. Server process using well known port crashes and restarts:

   All messages (of whatever sequence number) will appear as new ones, but the replies to clients will be detected as incorrect since they will carry "old" sequence numbers.

7. Server host crashes and reboots:

   In general servers will have new ports, but systematically started servers will resemble the last case.

8. Servers using well known port when host crashes and reboots:

   Servers will treat all clients as new, but clients will detect this from "old" sequence numbers in reply messages.

Figure 35 shows the rules for nesting calls and replies:

```
If (client and (no outstanding calls)) ->
       generate new conversation id
else (server or (nested call)) {
       use current conversation id stack expected reply information
       make call
       while is_call (listen on reply port (user blocked)) {
           hand to user
           {
                           who may make further calls
                           with same conversation id
           }
       reply } assert is_reply return to user
}
```

Figure 35. ESP calls and reply nesting rules

3.14 Related Protocols

Braden outlined the requirement for a standard protocol to support the optimal "two packet exchange" for local occasional exchanges, and yet still have the full functionality of an end-to-end WAN transport protocol.[Bra85a] The best known protocol that answers to this requirement is VMTP.[Che86a] Another early, but simpler design is described by Miller[Mil85b] The important distinctions between LAN specific and general WAN requirements for transport protocols are described in Danthine.[Dan80a] Finally, the classic paper on the specifics of transport protocols for Remote Procedure Call (but without much consideration for WAN characteristics) is in Birrell and Nelson.[Bir84a] Nor does this address the stack support required *inside* the protocol for callback.

ESP provides a good intermediate design for a working protocol to support RPC that meets Braden's requirements, and does not provide a large variety of extra features which cannot yet be justified by experiences (or else could be built at a higher level - e.g. the VMTP call forwarding facility).

4. Chapter 4: A Multicast Transport Protocol

This chapter presents the design of a reliable multicast transport protocol.[16] The aim of the protocol is to provide a service equivalent to a sequence of reliable sequential unicasts between a client and a number of servers, whilst using the broadcast nature of some networks to reduce both the number of packets transmitted and the overall time needed to collect replies.[17]

Most research into multicast services has been concerned with the network service, and mechanisms for routing multi-destination packets. This has often been in support of applications that need only an unreliable service (e.g. location services, audio and video). However, there are a variety of applications that need a reliable multi-destination delivery service, such as replicated databases and shared applications ("groupware"). In accordance with the end-to-end argument for system design, we believe this reliability should be introduced largely in the transport layer.

The service interface of the protocol offers several types of service, ranging from the collection of a single reply from any one of a set of servers to the collection of all replies from all known servers. The messages may be of effectively arbitrary size, and the number of servers may be quite large. To support this service over real networks, special flow control mechanisms are used to avoid multiple replies over-running the client. Reliable delivery is ensured using timeouts and a distributed acknowledgement scheme. The protocol is implemented over a network layer which supports multicast destination addressing and packet delivery. The behaviour of the protocol over both LANs and LANs interconnected by WAN lines is discussed. We also include some notions for possible future support from network interface hardware.

4.1 Introduction

This protocol supports a sequence of exchanges of arbitrarily sized request and response messages between a client and a large number of servers.**Cro85a, Croa, Cro88a** It is intended to replace existing protocols which employ either sequential unicasts or broadcast. One of the most important uses is to support replicated procedure calls**Bir84a, Wil87a** but it would also be appropriate as an underlying communications layer for replicated database access, for dissemination of information such as routing tables or electronic mail, or for location of services such as nameservers or gateways.**Wil87b**

The aim of this chapter is solely to present and evaluate the design of the protocol. We do not discuss, in any detail, the applications that might use multicast, nor do we discuss the management of multicast groups at the network or host level.

The first section of the chapter examines the range of multicast semantics required by distributed applications to be supported by the protocol. The next section describes the underlying network layer, together with its support for multicast destination addressing and packet delivery. The next two sections present the messaging service interface and operation. This sublayer of the protocol is designed to ensure reliable delivery of multicasts using a distributed acknowledgement scheme and retransmission after timeouts. It also provides multiple source and destination flow control using a coupled window scheme. The next section describes the request and response sublayer, and the use of *up calls* to provide *Voting* on replies. In the last two sections, we discuss a pilot implementation of the protocol and present some conclusions about the operation of the protocol.

We also include a formal analysis of the effect of many reply packets to a multicast request, the state required for the protocol and the packet formats used in the pilot implementation. We also discuss what kind of hardware support might make the protocol implementation simpler.

---

16. *"Many rivers to cross, and I've still got to find my way back home"*, Jimmy Cliff in The Harder the Fall.

17. Appendix 1 examines application needs for multicast, and presents the initial design of a real time window based Conferencing system. Appendix 2 examines mapping the window protocol onto another transport service.

4.2 Multicast Semantics

A range of multicast semantics and how they relate to the application requirements have been discussed in the literature.[Bir87a, Che85a, Hug86a] Here, we are only concerned with the transport service semantics. We assume that group membership is managed by some other mechanism, and that it can change during a single exchange. However, we allow the user of this multicast protocol to state the group membership, and *lock onto it* for the duration of each single exchange. By this we mean that once a server is replying, the group management mechanisms will not allow it to leave the group, or that if it does, a failure will be reported by this protocol, since the server process involved will no longer be correctly addressable. This will involve interaction between the application, the protocol and the group management mechanisms.

This protocol is geared towards client-server type applications, in which a client requests a service from a group of which it is not a member. However, it is possible to use the protocol in peer-to-peer type applications, where members of a group communicate amongst themselves. In this case, each member will also receive its own multicast requests. The protocol supports a variety of *types of service*, ranging from the collection of a single reply from any of a set of (possibly identical) servers, to the collection of all replies from all known servers. On receiving the initial part of a reply, the protocol will lock onto this given server, and may cancel the request to other servers. When more than one, or all the replies are required, the protocol will periodically retransmit the request until it has received all the replies. Failure modes are exactly analogous to those of a sequence of reliable unicasts, one to each server.

If a multicast request only requires replies from a subset of the group addressed, and this subset is indicated by the user (either by size, or by a list of specific addresses) then the client will lock onto servers as they reply. The protocol will inform non-members of the subset that their replies are not required. This is necessary for performance reasons, since dynamic regrouping into sub-groups could involve too many interactions with the group management system.

4.3 N-Reliable Maximal Service

The protocol presented here is intended to support *n-reliable* multicast requests and replies, where *n* denotes the number of servers guaranteed to see the request. In most cases this will correspond with the number of replies required. The client may specify *how many* and in *which members* of a group it is interested, where just *how many* may be wild-carded meaning *all current members* and *which members* is specified in a list of zero or more normal unicast addresses, indicating a subset of the multicast group of servers.

Many applications require only *1-reliable* multicast. These are usually *location* type services, where the exchange is of the form:

- Multicast Req: Where is Service X?
- N Unicast Replies: Service X is at Y.

Any reliable transport protocol may be enhanced to support this by simply delaying binding the destination address to a particular member of a multicast group, until some satisfactory part of the reply message or connection setup (if stream oriented) has been received from any or a particular member. (This may however have consequences for the rate at which sequence numbers can be recycled).

For applications requiring n-reliable multicast, where n is more than one, the user may also specify a *voting function*, which is *up-called* from the receiving code when one reply has been received from each of the (possibly individually specified) servers. An example of an application requiring this service might be failure recovery in a replicated system, where voting on the replies would be used for a re-build request - the vote function might be based on timestamps. So if the user specifies 1 out of any of the servers, with no voting, we have the weakest semantics. If the user specifies n, "out of all", and no voting, we have the equivalent of sequential unicast requests (with perhaps less chance of the group changing). If the user specifies voting, the protocol will return some single reply, or subset of replies based on the user supplied voting function.[18]

In this sense, as far as the receipt and *reaction* to a multicast request is concerned, we may call the protocol "At most once - At least all", where the first part of the description refers to the the call semantics, and the second part to the number-of-replies required for success. As far as the successful operation of the protocol is concerned in packet delivery, we may call the protocol "At least once - At most all".

4.4 Underlying Network Service

We assume that the underlying network service has three characteristics: It provides a datagram service such as the ISO connectionless network service or the DoD Internet Datagram Protocol. Host multicast or *host group* addressing is supported, and where possible, a packet addressed to a group address will use a physical broadcast or multicast facility on the network (e.g. on broadcast LANs or Satellite networks). It is possible, by some out of band mechanism, to ascertain the individual addresses of "exactly all" the members of a group, if required. This may itself be an application using some form of this multicast protocol.

The assumption of a physical broadcast facility obviously does not hold for most WANs and some non-broadcast LANs. For these types of networks, Deering[Dee85a] suggests a collection of *multicast agents* to forward multicast packets over point-to-point links, with no change to the best attempt datagram protocol provided by the Internet.

The agents for WAN multidestination delivery will usually form some spanning tree to route the packets.[Dal70a] The client is unlikely to be at the root of such a tree, in which case the route that reply packets follow has more bottlenecks than just that at the client. (There may be some part of the network that is far slower than any actual client host). The operation of an adaptive retransmission timer and the aggregate worst case window scheme in the transport protocol will alleviate the problem of packets imploding at such a bottleneck.

The reliable delivery guaranteed by some connection oriented (e.g. X.25) networks is not necessarily an advantage to our multicast transport: When the multicast group is large, it is possible that the agents, responsible for multidestination packet delivery at the network level, may not support adequate network level connections to reach all destinations at once. Rather than enhance them to buffer all the packets, and round robin deliver through connections as they become available, it may be wiser to leave this functionality in the transport level.[Sal84a]

Thus we see that the transport should be similar whatever the underlying service, and there is little point in adding more than a multidestination addressing and routing capability to any underlying network service.

4.5 The Messaging Service Interface

Given that a datagram service is not reliable, we must introduce some acknowledgement, timeout and retransmission scheme to ensure delivery of a message to the required number of destinations. message.[Che86a, Cro85a, Pan85a] We must also deal with the fact that large messages have to be broken into segments, which may then be lost, duplicated and misordered. These problems are solved by the messaging service and are offered as facilities by the service interface at this level.

The messaging service is in effect a simplex data protocol, which is then used by another layer to associate a request with a number of responses.

---

18. If the request is to a replicated service (e.g. database server) and the replies are different (due to network partition followed by partial recovery for instance), the client may wish one copy of the *latest* reply, and a copy of all the older replies so that it can perform some as yet unspecified application level recovery. An example of an application requiring all the answers would be the use of this to provide transport for a routing update protocol. A router would request an update from all of its neighbouring routers, and require all the answers, whether they are the same or different to be able to perform a calculation to form a consistent forwarding database.

4.6 Message Primitives

There are two varieties of send primitive, and two varieties of receive primitive. All message primitives may take an arbitrary size buffer of data, and reliably deliver it once (or indicate failure) to each destination. There is a mechanism to abort the sending of any message, which results in an error report being sent to the receiver. The primitives are:

- CMSend(Mid, Buf)

  This is for a user to send a multicast message (Buf) to a group identified by Mid, a multicast identifier.

  When the client calls CMSend, it is effectively blocked until everything has been sent and acknowledged (except for the last window's worth of packets in the message: see section 5.1).

- CMRecv(Mid, IdList, BufList, VoteFn, Timeout)

  This is for a user to receive a list of replies (BufList) from a list of possibly specified members of the group (Mid, IdList). IdList may be a wild-card to mean *any n*. VoteFn is the (optionally) user supplied function to vote on received messages.

  When the client calls CMRecv it is blocked again until all the packets from *all specified servers* have arrived, or the timeout expires.

- SMSend(Id, Buf)

  This is intended for a server to reply and looks like a simple reliable unicast message.

- SMRecv(Id, Buf, Timeout)

  This is for the server to receive a request.

- Abort(IdList)

  This cancels any outstanding messages to the list of receivers.

Applications can choose the weakest semantics they can use, to achieve the maximum performance.

4.7 Buffering and Markers in the Interface

The buffer parameters to the service interface are not contiguous bytes in memory. Buffers are linked lists of descriptors which hold pointers to sections of contiguous memory, together with the length of that section of memory. This avoids the need for a presentation layer type conversion to place the network format values in a contiguous buffer. Where the host presentation format is the same as the network format, or of the same size, this means that no buffer copying is required at all. In the latter case, type conversions may be done in place where necessary.

Each message carries an end of message marker in the last packet, and the protocol guarantees that boundaries indicated by the sender's marks will be preserved in calls to the receiver routines. Therefore the (possibly automatically generated) presentation layer does not have to check for application object boundaries overlapping buffer boundaries.

4.8 Failure Modes

CMSend can fail if the number of retransmissions of any part of the multicast message exceeds some threshold. CMRecv can fail if the requested number of replies are not forthcoming inside the timeout. SMSend and SMRecv fail in similar ways.

When a multipacket multicast transport protocol operates over an internet, it is susceptible to more complex failure modes of the underlying network. The protocol should make use of network reachability information, especially if this changes during a CMSend or CMRecv. If a message is received from any agent or router that a requested member, or more than an acceptable number of requested members, are not reachable, the protocol pre-empts the timeout and reports failure to the user.

4.9 Messaging Protocol Operation

CMSend and SMSend cut large messages into packets for transmission over the network. CMSend uses the multicast network service to transmit packets. SMSend is essentially a unicast version of CMSend with simpler acknowledgement and retransmission schemes.

Both choose a packet size appropriate to the worst case network route (from a routing table if possible). All packets carry a packet sequence number, and this is used by the receivers to filter duplicates, to sort misordered packets and to detect missing packets. The last packet carries a flag to indicate the end of a message.

4.10 Acknowledgement Scheme for Multipacket Messages

The acknowledgement scheme used at the message level depends on the number of transmitters and receivers. Acknowledgements of a multicast message are unicast by *each* of the receivers to the transmitter, rather than multicast to the group, to minimise the number of packets on an internet and minimise host processing on a local area network.

Acknowledgements also carry the receivers' advertised window. The client generally adapts its transmission rate to smallest window (see section 5.4). However, lagging servers may selectively acknowledge out of sequence packets, for the client to unicast missing packets. Since the higher layer of the protocol may send a reply, the protocol will piggyback acknowledgements to the last windows worth of packets in a message on the next message in the opposite direction. A timeout operates to trigger an *explicit* acknowledgement if a further message is not generated soon enough.

At one point we considered using a rotating primary receiver scheme based on a design by Chang and Maxemchuk.**Cha83a** This involves designating a special server (the primary), which multicasts acknowledgements to the current request (the primary moves to another server for the next packet). If a secondary server sees the multicast acknowledgement for a packet sequence number greater than the last one it received, it can trigger a negative acknowledgement to the client, who then (unicast) retransmits the missing packets.

We now think that this is inappropriate on LANs for the following reason: Since this protocol supports the exchange of large messages, the number of multicast acknowledgements would be large, causing more work for the group. This is very undesirable when the underlying multicast delivery in the network is not based on broadcast (i.e. agents in the internet).

The window and selective retransmission scheme operated by the client part of this protocol have the same functional effect without the overhead.

4.11 Retransmission Schemes

Retransmissions are necessary if a packet remains unacknowledged for a set period of time.

CMSend uses a dynamic retransmission timer in the same fashion as TCP.**Pos80a** This timer is set as the upper bound of the round trip times estimated for each destination in the multicast group. When the retransmission timer fires in CMSend, the client calculates the proportion of hosts which have not yet acknowledged the outstanding packets.

If this proportion is beyond some threshold, CMSend *multicasts* the retransmission. Otherwise we can optimise the retransmission scheme by unicasting the packet only to those hosts which did not send acknowledgements. The threshold is set depending on the distance and number of the servers, lower for LANs and higher for internet use. This is done to minimise timer events in the client and packets on a local network.

The last packet of a message is retransmitted periodically to check for client/server liveness with a bit set to indicate that this is deliberate.

SMSend uses a timer similar to that in CMSend, but based on the single server-client path.

4.12 Duplicate Filtering

Each packet carries a 32 bit sequence number as well as source and destination identifiers. Duplicate packets are discarded but acknowledged, in case the duplicate arose from a lost acknowledgement, and also to allow the transmitter to keep an up-to-date round trip time measurement.

4.13 Window Scheme

A window scheme operates on the segments of a message both to limit flow and to utilise long delay networks effectively. The scheme operates in two different ways:

1. CMSend operates an overall transmit window for the group and a separate transmit window for each of the servers.

   The overall window is the lower bound of the smallest of the advertised receive windows and the estimated number of packets that the pipeline with the shortest delay times bandwidth path to a server will hold.

   The separate windows are used to limit the number of unicast retransmitted packets to any server.

2. CMRecv keeps a receive window for each server, plus an overall receive window for the group. When the servers start to use SMSend, the client allows some of the servers to send some small number of packets more than others. The protocol closes individual windows if one server is too far ahead of the others in a reply or if the aggregate reply window is too large.

   This is to enable a host receiving many replies to pass parts of each reply, *in order* and also *in step*, up to the next level.[19]

4.14 The Implosion Problem

A common problem with distributed systems is the tendency of collections of related events to synchronise. In particular, if servers responding to a multicast request do so nearly all at the same time, they may swamp both the network and the client, causing a high number of packets to be dropped. If they then time out and retransmit at nearly the same time the same problem will occur again, leading to very poor overall performance. This is illustrated in Figure 36.

19. The problem of how to adapt to the receiver group membership changing during an exchange is a topic for future research.

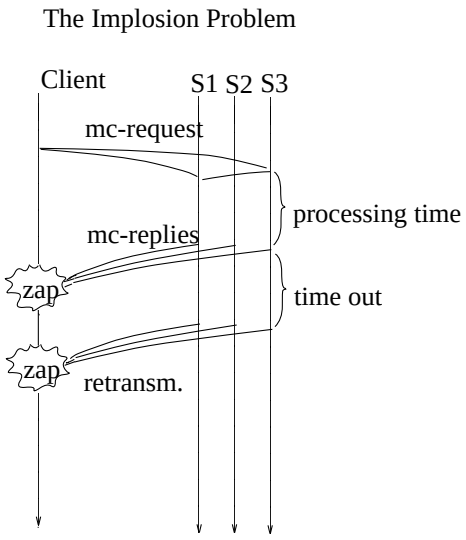


Figure 36. Multicast Response Implosion

Statistical analysis of this problem on an Ethernet shows that the limiting capacity is likely to be the client rather than the network. This analysis attempts to quantify what a *large* group size is for Ethernet operation of the protocol, so that we may choose appropriate values for timers and windows.**Fra85a, Ram87a**

In the analysis that follows, there are a number of simplifying assumptions that affect the accuracy of the results, although not the basic trends. In particular, the analysis of the implosion problem in 4.21 assumes the following:

1. The time to send a packet is the same as the time to decide whether to send a packet (aka contention slot). This could actually be enforced, but to be realistic, further analysis should be done to discover the affect of the real difference.

2. The probability replies not being ready, and thus not using a "slot" and the probablility of replies colliding remain constant throughout the exchange. In fact, as more servers reply, the chance of not using a slot may increase, while the chance of colliding will decrease. This may cancel to first order.

Suppose that a multicast request has been received correctly by n servers, all at the same time t. Each of them, will try to *acquire the ether* or access the network during the time interval $[t, t+\delta t]$ with probability α. (This is equivalent to assuming that the servers respond with some kind of randomised delay). If one of the servers succeeds in acquiring the ether, the others wait until it finishes transmitting, at a time t_2 say. They then compete during the next interval, $[t_2, t_2+\delta t]$, in the same way as they did for the first. If, on the other hand, there was a collision during the first contention interval, all servers involved in the collision stop transmitting at once, and repeat the contention process at time $t+\delta t$. We first assume that background traffic is so light as to be negligible. We also assume that the replies are of similar size.

With these assumptions we can calculate the mean number of contention slots needed to transmit n replies, as well as the mean number of packets needed overall.

Values for α depend partly on the number of servers involved and partly on the length of the packets they are trying to send: the length of contention slots in the Ethernet is of the order of *Contend* μs. Assume that typical reply times after a receipt of a multicast lie in the range *Range* μs. The actual values will be determined by experiment.

Since we are in the downward sloping part of the curves, collisions and hence retransmissions are rare and can be neglected for this calculation.

Assuming furthermore that the reply times are evenly distributed, then the probability of an attempted reply in any contention slot can be taken as approximately

$$\alpha = \frac{Contend}{(Range - nl/Contend)}$$

where n is the no of servers and l is the length of their replies in bits.

Medium or high levels of background traffic during a multicast transaction can be modelled as follows: we denote the probability that any host not involved in the multicast transaction will transmit during a current contention slot as β. Substituting and evaluating for different values of β seems to show that β affects mainly the number of slots needed: the number of packets needed increases only at very high values.

4.15 Scheduling replies

A client may able to accept the aggregate rate of reply from some number of servers. However, if the servers' reply packets tend to clump together, then the client may well be swamped by back-to-back packets.

A good implementation of the selective unicast acknowledgement scheme used by this protocol will randomise the scheduling of replies from each of the servers. In the literature, this problem has been solved by introducing independent random back-off schemes at each server.**Dee85a** This has the disadvantage that it cannot be adapted dynamically: reply times will be delayed even if changing circumstances make it unnecessary. This can be overcome by driving the scheduling scheme from the client.

A simple algorithms for this is to take the current aggregate window and divide it evenly over all the servers. A receiver then schedules acknowledgements round robin to each server with a mean and variance of delay based on the time for processing one server's window worth of packets.

In the local network case, this may not be feasible, since the processing times per packet need to be fast, and the variance on packet transmission times by each server tends to be too low for such a complex statistical calculation to be done dynamically.

4.16 Request and Response

The reliable message protocol can be used by a request response protocol.

This has five service primitives:

1. Request(Mid, IdList, Buf, Flags, Timeout)

   This is used by the client to send a part of a request. Mid is the group identifier. IdList is a possible subset. Buf contains some of the request data. Flags indicates the service wanted and an indication if this is the last part of the request.

   Timeout here is set by a fault tolerant application, which may give up a request after some time. Normally it would be infinite.

2. Response(IdList, BufList, VoteFn, Flags, Timeout)

   This is used by the client to collect reply messages. Flags is used to indicate the type of multicast receive service. Depending how it is set, Response can return with each buffer from each server, or with the list of buffers (if servers are deemed equivalent) for this part of the message, or with all the entire messages. Timeout allows the user to poll for replies.

   An indication in Flags is set for the end of a reply.

3. ReqAbort(IdList)

   This allows the user to abort a request to a given number of servers. It results in an error packet being sent to each of those servers.

   Request, Response and Abort are used together by a client. The might be part of an automatically generated stub for a remote procedure call.

4. Listen(Id, Buf, Timeout)

This is used by the server to listen for calls.

5. Reply(Buf)

   This is used by the server to send a reply.

   Listen and Reply are used by a server, and may be used as the counterpart of the client's stub described above.

In addition to these functions which build fairly straightforwardly on the message level, the first packet of a reply message can be used to acknowledge delivery of a request to that server, and a new request from the client to the server can be taken to indicate that the client has received *all* the replies that it is interested in, and that servers may discard any held replies.

Requests and responses carry a conversation identifier to associate them, and a special bit to distinguish them.

4.17 Voting Functions

The voting function is provided by the user to the request-response level. Most frequently, this is a function that takes a list of buffer descriptors and simply compares some particular field to some value. It may also compare the list of replies against each other and return the most common.

4.18 Operation of Request/Response using Message Level

The request-response level uses the message level. This operates as follows:

- Request sets up state for the request, and uses CMSend to issue each part of the request.
- Response uses the state from Request, and calls CMRecv to collect appropriate parts of replies from servers. When unwanted replies arrive at a client, CMRecv may call Abort so that the Client sends error reports to save servers unwanted work. Similarly, if the VoteFn has been applied successfully, Abort will be called for any remaining servers. Each call to Response hands the equivalent segments of replies from different servers to the user.
- ReqAbort is called by the user when some out of band event means that the Request is no longer useful. It uses Abort to cancel all the servers' effort.
- Listen uses SMRecv to collect each part of a request from a client. It saves the source address of the request.

  Reply simply uses SMSend to send each part of the reply message to the saved address.

4.19 Failure Modes and Cancelled Requests

The failure modes of the request response layer are derived from the failure modes of the message layer in the obvious manner.

If the client aborts a request while receiving a reply, the protocol sends a message level abort to all the servers in the group. This is to enable the servers to tidy up any state and discard the replies they may have buffered.

4.20 Implementation

There is an experimental implementation of this protocol in *user space* under Berkeley Unix. It is similar in some ways to the ones designed in**Hug88a, Liu86a**

Initial use and performance of this protocol on a single LAN is under investigation. For a single packet call and reply, initial performance against number of servers behaves as one would expect from the theoretical analysis. We have not yet experimented with multipacket calls and replies, either on a single LAN or on an Internet.

4.21 Analysis of Implosion for Single LAN

This analysis starts from a method suggested by the author. The development is due to Karen Paliwoda with some assistance from Dr Paul Otto.

We make the assumptions presented above.

Let $p_{i,i-1}$ be the probability that out of i receivers still trying to respond, one is successful during the current contention slot (thus reducing the number of outstanding responses to $i-1$). This will be the case if only one tries to transmit while all the others are silent:

$$p_{i,i-1} = i\alpha(1-\alpha)^{i-1} \qquad \textit{where} \quad 0<i\leq n.$$

Let p_{ii} be the probability that no message is transmitted during the current contention interval (either because no receiver tried to transmit or because of a collision).

$$p_{ii} = 1 - i\alpha(1-\alpha)^{i-1}$$

Then the probability $Pr(n)$ that all n messages are transmitted using only n contention slots is the probability of a successful transmission following every slot:

$$Pr(n) = p_{n,n-1}\, p_{n-1,n-2} \cdots p_{21}\, p_{10} = \prod_{i=1}^{n} p_{i,i-1}$$

The probability $Pr(n+1)$ that $(n+1)$ slots are needed can be calculated as follows: out of $n+1$ slots n must have been used for successful transmissions whereas the remaining one was badly used in the sense that either it went by unused or it contained a collision. This badly used slot may have been the first when all n receivers still had to respond, or the second, with $(n-1)$ receivers still trying to respond, or indeed any of the others. Hence:

$$\begin{aligned} Pr(n+1) &= p_{nn}\prod_{i=1}^{n} p_{i,i-1} + p_{n-1,n-1}\prod_{i=1}^{n} p_{i,i-1} + \cdots + p_{11}\prod_{i=1}^{n} p_{i,i-1} \\ &= \sum_{i=1}^{n} p_{ii} \prod_{i=1}^{n} p_{i,i-1} \end{aligned}$$

Similarly, it can be shown that:

$$\begin{aligned} Pr(n+2) = {} & [\, p_{nn}(p_{11}+p_{22}+\cdots+p_{nn}) \\ & + p_{n-1,n-1}(p_{11}+p_{22}+\ldots.+p_{n-1,n-1}) \\ & + \cdots \\ & + p_{22}(p_{11}+p_{22}) + p_{11}^{2}\,] \prod_{i=1}^{n} p_{i,i-1} \\ = {} & \Big(\sum_{i=1}^{n} p_{ii} \sum_{j=1}^{i} p_{jj}\Big) \prod_{i=1}^{n} p_{i,i-1} \end{aligned}$$

and:

$$Pr(n+3) = \Big(\sum_{i=1}^{n} p_{ii} \sum_{j=1}^{i} p_{jj} \sum_{k=1}^{j} p_{kk}\Big) \prod_{i=1}^{n} p_{i,i-1}$$

In general, these and further expressions can be obtained by using a lower triangular matrix P and vectors $\underline{p}$ and $\underline{u}$:

$$P = \begin{bmatrix} p_{11} & 0 & 0 & \cdots & 0 \\ p_{22} & p_{22} & 0 & & 0 \\ p_{33} & p_{33} & p_{33} & & 0 \\ . & & . & & . \\ . & & . & & . \\ . & & . & & . \\ p_{nn} & p_{nn} & p_{nn} & \cdots & p_{nn} \end{bmatrix}, \quad \underline{p} = \begin{bmatrix} p_{11} \\ p_{22} \\ p_{33} \\ . \\ . \\ . \\ p_{nn} \end{bmatrix} \quad \text{and} \quad \underline{u} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ . \\ . \\ . \\ 1 \end{bmatrix}$$

Using the dot product of $\underline{p}$ and $\underline{u}$, we can write:

$$Pr(n+1) = \prod_{i=1}^{n} p_{i,i-1} \, (\underline{p}.\underline{u})$$

$$Pr(n+2) = \prod_{i=1}^{n} p_{i,i-1} \, (P\underline{p}) . \underline{u}$$

and, in general,

$$Pr(n+i) = \prod_{i=1}^{n} p_{i,i-1} \, (P^{i-1}\underline{p}) . \underline{u}$$

Then the mean number of slots needed is μ_S:

$$\mu_s = \sum_{i=0}^{\infty} (n+i) Pr(n+i)$$
$$= (n + (\sum_{i=1}^{\infty} (n+i) P^{i-1}) \underline{p}.\underline{u}) \prod_{i=1}^{n} p_{i,i-1}$$

Now P can be diagonalised as $Q \Lambda Q^{-1}$, where $\Lambda = diag(\lambda_i)$, the diagonal matrix with the eigenvalues of P on the diagonal, and Q is the matrix whose columns are the eigenvectors of P, Q^{-1} being the inverse of Q.

We then have :

$$\mu_S = [n + (\sum_{i=1}^{\infty} (n+i) (Q \Lambda Q^{-1})^{i-1}) \underline{p}.\underline{u}] \prod_{i=1}^{n} p_{i,i-1}$$
$$= [n + Q (\sum_{i=1}^{\infty} (n+i) \Lambda^{i-1}) Q^{-1} \underline{p}.\underline{u}] \prod_{i=1}^{n} p_{i,i-1}$$

$$= [n + Q \; diag \; (n \sum_{j=1}^{\infty} \lambda_i^{j-1} + \sum_{j=1}^{\infty} i \; \lambda_i^{j-1}) Q^{-1} \underline{p}.\underline{u}] \prod_{i=1}^{n} p_{i,i-1}$$

The eigenvalues λ_i of P are $p_{11} p_{22} \cdots p_{nn}$, and hence $|\lambda_i| < 1$, for all i.

Therefore the series

$$(n \sum_{j=1}^{\infty} \lambda_i^{j-1} + \sum_{j=1}^{\infty} i \; \lambda_i^{j-1}) = \frac{n}{1-p_{ii}} + \frac{1}{(1-p_{ii})^2} = \frac{np_{i,i-1}+1}{p_{i,i-1}^2}$$

It can then be shown that

$$Q \; diag(d_i) Q^{-1} \underline{p}.\underline{u} = \sum_{i=1}^{n} \frac{p_{ii}^n d_i}{\prod_{j=1, j \neq i}^{n} (p_{ii} - p_{jj})} \quad \text{where} \quad d_i = \frac{np_{i,i-1}+1}{p_{i,i-1}^2}$$

Note that this formula cannot be applied if any $p_{i,i}$, $p_{j,j}$ are equal for any different i, j. In this case, it is impossible to diagonalise the matrix P. However, because the spectral radius of P is less than one, μ_s is still guaranteed to be finite.[20]

The mean no. of packet transmissions can now be obtained as follows: out of μ_S contention slots needed to transmit all replies, we know that there were n slots in which a transmission was successful, incurring one packet each. Out of the remaining $(\mu_S - n)$ slots, a proportion were idle slots, and the rest involved collisions.

The probability of i receivers being silent during a current contention slot is $(1-\alpha)^i$. So at a stage where i receivers still have to respond, the probability of them being silent given that there was no packet transmitted during the current slot is:

$$\frac{(1-\alpha)^i}{1 - i\,\alpha(1-\alpha)^{i-1}}$$

Over all stages, this averages to:

$$s = \frac{1}{n}\sum_{i=1}^{n}\frac{(1-\alpha)^i}{1 - i\,\alpha(1-\alpha)^{i-1}}$$

The probability of collisions is $1-s$.

Trivially, the mean number of packets transmitted in an unused contention slot is 0.

The mean number of packets involved in a collision at a stage when i receivers still have to respond is:

$$Col_i = \sum_{j=2}^{i} j\binom{i}{j}\alpha^j(1-\alpha)^{i-j} \qquad where \quad i \geq 2$$

Hence over all stages the mean number of packets per collision is: $ppc = \frac{1}{n}\sum_{i=2}^{n} Col_i$.

So the mean number of packets sent in μ_S contention slots is μ_P:

$$\mu_P = n + (\mu_S - n)s * 0 + (\mu_S - n)(1-s)ppc = n + (\mu_S - n)(1-s)ppc$$

4.22 Differing Background Loads

Medium or high levels of background traffic during a multicast transaction can be modelled as follows: we denote the probability that any host not involved in the multicast transaction will transmit during a current contention slot as β. The probability of a successful response to the multicast then changes to

$$p_{i,i-1} = i\,\alpha(1-\alpha)^{i-1}(1-\beta) \qquad where \quad 0 < i \leq n.$$

Consequently

$$p_{ii} = 1 - i\,\alpha\,(1-\alpha)^{i-1}(1-\beta)$$

Substituting these new values into μ_S and μ_P and evaluating for different values of β seems to show that β affects mainly the number of slots needed: the number of packets needed increases only at very high values.

4.23 Potential Hardware Support

For large multicast requests, it would be convenient to have network interfaces that filtered not only on network address, broadcast and multicast address, but also on sequence number of packets within messages. This would obviate the need for a complex selective retransmission scheme.

---

20. Thanks to Paul Otto for pointing this out.

Multicast protocols are most attractive where the underlying technology is broadcast. One of the main problems on a broadcast medium is the excess work for hosts not interested in a current broadcast packet. Current hardware support for *filtering* packets is based simply on per host multicast address lists.

A convenient extension of this would be to filter on packet source and sequence number. The Cambridge ring minipacket mechanism[JNT82a] provided the former, but no LAN or WAN interfaces provide sequence number filtering. Mockapetris suggests a scheme for a pseudo "alternating bit protocol" to filter unwanted multicast packets.[Moc83a] Mogul, Rashid and Accetta[Mog87a] suggest a more general but similar mechanism within the operating system software to support efficient user level protocol implementation.

This could be extended down to hardware, to use multiple filters per multicast address. Hosts would accept all packets within some sequence space addressed to them, and the hardware would roll the sequence number filter forward as each packet was successfully passed up to the application.

4.24 Protocol State Required

The protocol state and packet formats are presented here in the language used to define interfaces for the RPC systems the protocol is designed to support. These definitions are definitive as they are machine readable, and can be processed to generate implementations of parts of the protocol. The text in the rest of this chapter should be read by an implementor as supporting explanatory material, but the specification here takes priority where any ambiguity or inconsistency is present. This is presented in Figure 37.

```
-- Multicast semantics

constant ANY_N_RELIABLE 1
constant KNOWN_N_RELIABLE 2
constant SOME_KNOWN_N_RELIABLE 4

constant WHICH_MASK 0x07

def Any_N(x)
      ((x->NReliable&WHICH_MASK)==ANY_N_RELIABLE)
def Known_N(x)
      ((x->NReliable&WHICH_MASK)==KNOWN_N_RELIABLE)
def SomeKnown_N(x)
      ((x->NReliable&WHICH_MASK)==SOME_KNOWN_N_RELIABLE)

constant LESS_N_RELIABLE 8
constant MAJORITY_RELIABLE 16

def Reliable(x)
      ((x->NReliable&HOW_MANY_MASK)==0)
def LessThanN(x)
      ((x->NReliable&HOW_MANY_MASK)==LESS_N_RELIABLE)
def Majority(x)
      ((x->NReliable&HOW_MANY_MASK)==MAJORITY_RELIABLE)

constant HOW_MANY_MASK 0x70

-- Acknowledgement Style

constant FLOOD_STYLE 0
constant MACK_STYLE 1
constant SACK_STYLE 2
constant NACK_STYLE 3


-- Per server info kept at client

type PerServer {
      Mid     Server;
```

```
        int     State;
        Seqno   PktSeq;         -- Seq no. ack'd by this server so far  --
        Window  RxWindow;       -- Window for this server               --
        Window  TxWindow;       -- Server's rx window                   --
        Pkt     *Rep;
-- Stats on Calls --
        int     Rtx;
        int     ReplyCount;
} PS;

-- Client state table holds all info for
-- this client & forall servers

type ClientState {
-- Addressing info
        Mid     Client;         -- My address
        Mid     GroupAddr;      -- Group Address
        Seqno   AllPktSeq;      -- Latest seqno over all
        Window  AllWindow;      -- Overall window on all svrs
-- Info required Per Server in the Group
        PS      *Member;
        int     Grouplen;       -- Length of group list
-- Fn supplied by user to vote on replies if thats whats wanted
        function  voter();      -- User supplied voting fn
-- Protocol Specific Info
        int     NReliable;      -- Type of multicast
        int     Style;          -- Ack style
        Tim     Timeout;        -- Curr Timeout for each member
        int     State;          -- Overall state
-- Request Info
        Seqno   This;           -- Request Sequence number
        Pkt     Req;            -- Current request
-- Stats on Calls
        int     TotalRtxs;      -- Count of all rtxs
-- Handy network things
        int     Socket;         -- Handle on network
        int     AddrSize;       -- sizeof a group addr
} CPCB;


-- State for each Member server in Group
-- Total state is some convolution of the member states

constant IDLE_STATE    0
constant REQ_STATE     1
constant REP_STATE     2
constant ACK_STATE     3
constant NACK_STATE    4

-- Overall state of call

constant PROGRESS_STATE  0
constant REPLIED_STATE   1
constant FAILED_STATE    2

constant OVERSUBSCRIBED 64

-- Server state = 1 of these foreach client + general info

type PerClient {
        Mid     Client;         -- Address of a past client
        Seqno   Last;           -- Last msg no. from that client
        Seqno   PktSeq;         -- Pkt in msg so far
        Window  Window;         -- Client's rx window for us
        int     State;          -- Our state for that Client
        Pkt     Rep;            -- Saved reply for that client
```

```
        PerClient *Next;
} PC;

type ServerState {
-- Address Info
        Mid     Server;         -- This Server
-- Per Client Info
        PC      *PerClient;
        int     ClientCount;
        Window  Window;         -- My current rx window
        Pkt     Req;            -- Current request
-- Protocol Info
        Tim     Timeout;        -- Timeout on Replies ??
        int     State;          -- state of server
-- Fn supplied by user to do server work
        function Work();
-- General things of use
        int     Socket;         -- Network handle
        int     AddrSize;       -- Length of addr
} SPCB;



-- Window Size base values

-- This is typically for a unicast multipacket msg
constant AGGREGATE_WINDOW       16

-- While this is a max for a fast replying server
-- to be ahead of slow ones
constant EACH_SERVER_WINDOW     4
```

Figure 37. Definitive Multicast Transport Protocol Specification

```
-- Packet structure

type p {
      Seqno   seq;    -- Req/Rep Seq matching
      char    type;   -- Sort of pkt
-- Per pkt info
      Seqno   pseq;   -- Pkt or Ack   within a
                  -- single Req/Rep
      Window  win;    -- Current advertised rx window
      short   len;    -- This pkt len in bytes
      short   flags;  -- indicates last pkt now
      Buffer  dat;
                  -- User Data
                  -- Also used for bitmap in SACKS
                  -- Sack pkt has list of holes,
                  -- Seqno + Len of each hole
} Pkt;

-- Packet Types

constant REQ    1
constant REP    2

-- Client to Server REP ACKS --
constant ACK    3
constant NACK   4
constant SACK   5

-- Server to Client Req Pkt ACKS --
constant PACK   6

constant LAST   1
constant MARK   2
```

Figure 38. Definitive Multicast Transport Packet Specification

4.25 Alternative Approaches - TCP Extension to One-to-Many Multicast

TCP was designed to operate in a network environment where packet delivery is not guaranteed. Packets may be lost, duplicated or delivered out of order as a result of buffer overflow, retransmissions and multi-pathing. TCP uses sequence numbers and retransmission to re-create at the destination an exact copy of the byte stream generated at the source. At the source, TCP assigns each byte of the data a sequence number and sorts the bytes at the destination according to their sequence numbers to eliminate any duplication and misordering. Lost and unacknowledged data are retransmitted when a retransmission timer expires. TCP provides a full duplex *One-to-One* Byte Stream.

There are a number of applications which have a requirement for a *One-to-Many* protocol, including:

1. Software Distribution in networked workstations
2. News Distribution is similar to the above, but perhaps requires the Wide Area facilities of IP Multicast Routing.
3. Conferencing applications often need to deliver the same audio or video to a collection of stations.

RFC-1112 **Dee85a** specified a mechanism for the management of groups of IP hosts, and a group addressing scheme. RFC-1075 **Wai88a** and the MOSPF IETF Draft **Moy91a** specify mechanisms for routing IP packets to these groups. These mechanisms do not, however, enhance the IP Protocol to avoid loss, re-ordering or duplication that can occur in the Internet. TCP is a protocol designed to do just that, but until now only for a one to one connection.

There are some applications which require *Many-to-One* connections:

1. Backup/Archival services for networked workstations
2. Distributed Transaction Processing
3. Error Logging

These would require a more complex extension to the TCP protocol (especially in the Application Programming Interface), which requires further study. Some of the requirements are captured in [5], but probably the most complete discussion may be found in [6].

In the rest of this section, we discuss how the TCP Protocol may be revised to operate using the IP Multicast extensions to achieve one to many connectivity. This represents an evolutionary approach to building a reliable multicast protocol, where the previous parts of this chapter have been from a revolutionary approach.

4.25.1 A Simple TCP Extension

A Client (Active in RFC 297 terminology) calls to a destination address (port + IP Multicast Group) from (one of) the host's normal IP address. The TCP process/task sends the SYN to that port, but to the multicast IP group address, having set up a single TCP Control Block (TCPCB).

Each Server (Passive) receives the SYN, and goes through the connection establishment procedure in exactly the normal way (RFC 792, Section 3.4), with one small change. When a TCPCB is in LISTEN state, and a SYN arrives, although it is to a destination IP address of the group, the returned packets are sent to the source IP address (not the group, of course) but also **from** (one of) the unicast IP addresses of the (each) server host. As SYN ACKs come back from members of the group (which can be deduced from the fact the SYN ACK's source address is the specific IP host address), TCP allocates more TCPCBs, each one accessible by the unique address.

Some mechanism for *chaining* the TCPCBs together is necessary, and linking them to the fact that they are in a group. We suggest (for implementation simplicity), that there is a sender TCPCB, with source address this host+this port, and destination address the group; then, chained from this are as many more TCPCBs as group members, with the unique IP address of each member. Since a TCPCB is typically about 128 bytes, this is not an expensive implementation mechanism. It can support further mechanisms discussed below.

The Client (Initiator) application calls some Application Programming Interface such as *accept* to accept the multiple connections, thus learning the source IP address membership of the group (and allowing ruling out of some members/voting) - this avoids having to change the *connect* system call.

Then to go on to the data communication phase, data is simply sent to multicast address + port. Returning data is dealt with as if it is in a unicast connection.

Acks/window schemes are a matter for future research.

To perform *read* operations in the API, we have to decide on a user interface. The following choices present themselves:

1. Read n* as much data for n servers...(i.e. rely on the application to provide a big enough buffer)
2. Use something analogous to readvec, each vector element gets loaded with reply from each server
3. If we care that the data may be different from different servers (e.g. voting etc), do a select on multiple sockets returned by the accept after the 1/2 connect phase...
4. prepend data in read (or iovec) with address of server (allows more arbitrary ordering)...

During Connection Establishment, Teardown, and Data Exchange, there is the question of what to do concerning packet losses. Here, we must make one strong assumption concerning the management of Multicast TCP Server Groups: *Any potential client knows the group membership in terms of the IP multicast group, and the unique IP addresses of hosts in the group*. This can be implemented by an out of band protocol between potential Clients and Multicast Agents, or else by configuration.

SYN, FIN loss or SYN ACK loss is then dealt with by retransmission of the SYN until connections are ESTABLISHED to the full group.

Data loss may be dealt with in a more complex way. Here we introduce the term *aperture*. This refers to the number of Server hosts in the group that we send any given packet to from the Client. In Normal circumstances this is *full open*. I.E. we are sending to the full group.

In the event of packet loss, there are 2 main cases.

1. The data packet was lost. Then we would like to send the packet to the group again.
2. Ack loss. Then would like to send to the individual member whose ack was lost.

This requires more mechanism in the TCP Protocol Engine. Recall that we have a TCPCB for every member. Retransmission is generally Client driven in any case, so we should have adequate information for most cases. However, the *paths* to the Servers are all potentially different, so there will be different Round Trip Time estimates for the different members. Thus it is not possible to completely determine whether the data to everyone got lost in a performant way. If the RTTs are sufficiently close, it may be reasonable to chose the highest as our basis for a retransmission timer. If they are significantly different, it may be reasonable to unicast retransmissions.[21]

4.25.2 Dynamic Window Adjustment

TCP has a complex dynamic window mechanism which is clearly more complex when operating over multiple paths. The simplest solution as to how to chose a send window when connected to multiple servers is to choose the smallest window offered (In fact we do not believe there is a choice).

In the event of loss, TCP implements congestion avoidance. Our approach to Multicast TCP so far has been to strongly couple all the Servers. However, if there is detectable packet loss to a single server (i.e. we see

21. If multicast groups were bit masks, we could naturally break them to subsets and do arbitrary retransmissions, but they do not have such structure, so this is not feasible.

acks from all the others), it may be feasible to implement congestion avoidance to that single site. However, at some point, the site will lag behind due to the number of round trip times that congestion avoidance takes to recover. At this point we therefore believe that the only choice is to apply the congestion avoidance to all the members of the group (possibly implemented by recording the worst case congestion window in the group TCPCB, and giving that precedence over the individual ones).

In any case, if there is congestion to one site, it is reasonably likely that there will be congestion to others too. This depends very much on the topological distribution of servers. For truly wide area distribution of news, this might not be appropriate.

4.25.3 Future Work

Future work is required to investigate the omission of a retransmission scheme, but inclusion of timestamping, to allow TCP to be used for real time applications such as voice and video conferencing. A great deal of tuning may be possible in the area of how the aperture for retransmission can be adjusted.

4.26 Conclusions

In this chapter we have presented the design of a complete reliable multicast transport service and protocol, suitable for replicated database access, or for implementing groupware applications. We have seen that these services are best achieved through a new protocol, but could also be provided by extensions to an exisiting protocol. Such a service is essential for timely, efficient operation of such applications. There are as yet no such services in any systems described in the current literature.

5. Chapter 5: Transport Protocol Performance

This chapter describes the tools and measurement mechanisms used to investigate the performance of the Transmission Control Protocol (TCP) over the Atlantic Packet Satellite Network (SATNET).[22] One of the main tools used was developed at UCL by Stephen Easterbrook at UCL, while the other was provided by Van Jacobson of LBL.

The relevance of this work is that similar dynamic mechanisms to those used by TCP are also in the design of both the Sequential Exchange Protocol and the Multicast protocol described in the last two chapters. It might be argued that TCP has largely been deployed successfully in workstations on LANs. However, historically it has been designed to function over a wider range of underlying media than any other protocol, and to this end is widely adaptable. The SATNET was a network at just one extreme. The bandwidth/delay product of the SATNET also means that it has dynamics which are very similar to higher speed terrestrial networks of the near future (128kbps * 1 second is roughly equivalent of the "pipesize" of gigabit LAN technology).

We assume the reader has a reasonable level of familiarity with protocol principles and implementations, although not necessarily with TCP itself. A good reference for measurement techniques is Treadwell's thesis.**Tre80a**

5.1 Introduction

Any complex protocol has three main points of interaction with the environment and these are where we first instrument any implementation. These are:

- The Service interface to the *user*.
- The interface to the layer below.
- The interface to a clock for timer events.

This can be envisaged as shown in Figure 39:

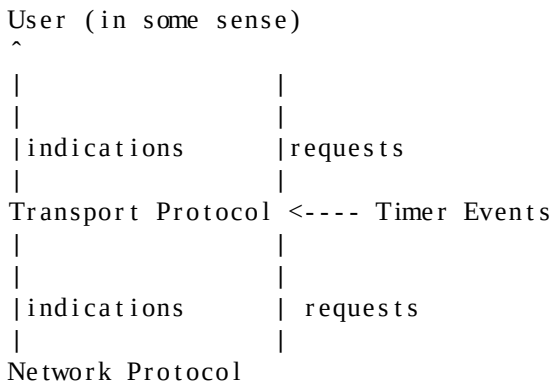


Figure 39. Event Trace Measurement Points

22. *The transport studies group at UCL reported in 1991 that the average speed of traffic on the London roads was now the same as in 1891: around 11 mph.*

In the case of the Transmission Control Protocol (**TCP**)[Pos80a] , these are easily identified as:

- The interface to the application (e.g. TELNET, FTP or SMTP).
- The interface between TCP and the Internet Datagram Protocol (**IP**).[Pos81a]
- The interface between TCP and the timer mechanism available with the protocol implementation environment, as shown in Figure 40.

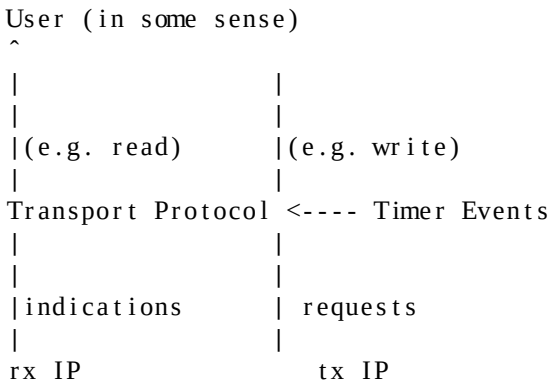


Figure 40. TCP Timer Events

A *reliable Transport Protocol* such as TCP is generally connection oriented, and thus contains a large amount of state information. This state information is held in a *Protocol Control Block* referred to in the TCP specification as a Transmission Control Protocol Control Block (**TCPCB**).

It is desirable to be able to access the TCPCB on any event occurring at each end of a connection, to check that the internal state of the two ends is consistent with the events (packets and user requests and indications) at each end. To achieve this is almost impossible when the two ends are widely separated, and our solution to this problem was to be able to correlate external events at each end with a local clock, and to access the TCPCB independently at each end. For the purposes of checking the correctness of the protocol implementation, this is adequate, but it has certain shortcomings when investigating performance. Typically a TCPCB contains the state variables shown in Figure 41:

```
General:
  State - i.e.  CLOSED, LISTEN, SYN_SENT, SYN_RECEIVED,
       ESTABLISHED, CLOSE_WAIT, FIN_WAIT1, CLOSING, LAST_ACK,
       FIN_WAIT2, TIME_WAIT... used conceptually
       with event to index into Finite State Machine
  Handle on user

Queues:
  Retransmit Queue
  Reassembly Queue

Send Sequence Variables:
  Send unacknowledged
  Send next
  Send window
  Send urgent pointer
  Segment sequence number used for last window update
  Segment acknowledgement number used for last window update
  Initial send sequence number

Receive Sequence Variables:
  Receive next
  Receive window
  Receive urgent pointer
  Initial receive sequence number

Timer Related:
  Idle timer
  Current retransmit timer
  no. duplicate acks
  Round trip timer
  Sequence number being timed
  Smoothed round trip timer
  Variance in round trip time (*)

Miscellaneous:
  Time to live to use
  Maximum segment size to use

Congestion Control: (*)
  Congestion-controlled window
  Snd_cwnd size threshold for slow start exponential
    to linear switch
```

Figure 41. TCP Protocol Control Block (TCPCB)

When the instruments to access events and the protocol state are in place, we need to exercise the protocol implementation in a controlled way. Firstly, we need a traffic source and sink that allows control of user requests. Secondly, we need a benchmark of how the underlying network behaves when used in the simplest possible way.

In the case of TCP/IP the first of these is relatively easy. A user may open, listen and close a connection, and send and receive numbers of bytes of data. Source and sink programs are therefore trivial, with the proviso that controlled timing of user requests to the protocol may not be feasible in many programming environments, since it may not be possible to trigger an event at an exact interval (due to preemptive scheduling or other non-determinisitc behaviour). In this case, we make sure that the traffic generation programs are *self clocking*.

The second of these may be harder to achieve, since the Internet is a non-trivial environment. Even direct use of IP provides wildly varying results in terms of throughput and delay distributions.

The most useful tools to achieve some characterisation of the underlying network without interference from Transport Protocol effects are based on the use of ICMP (Internet Control Message Protocol)**Pos80b** traffic generators, often using ICMP Timestamp requests to elicit round trip time information.

5.2 IP/ICMP traffic generation

UCL obtained the ICMP echo generator developed by Muus while at BRL, for BSD Unix, and modified it to generate echo requests at a measured rate. With a user space program under a multiuser operating system, a requested rate is not guaranteed. Instead, the program attempts to send requests at a given rate, measures what rate it actually achieves and adjusts upwards or downwards accordingly. Most machines that run BSD Unix have a poor clock resolution (typically +/- 10 msecs), even though reading a hardware clock should be feasible from user space without descheduling a process. However, for the networks for which we generate traffic, this allowed possible interpacket times far in excess of the bandwidth the network could handle, and was therefore adequate.

Other parameters that can be set are:

- The size of the packet (including ICMP and IP headers).
- The timeout to wait before replies.
- The number of requests to send in a single run.
- Whether the destination is an IP echo site (and therefore to choose to send ECHO replies to avoid a 4 way trip).
- The source address to use if the host is multi-homed.

The ICMP traffic generator counts the packets out, and counts the replies (matching by ICMP sequence). To allow decisions about lost packets, we set a timeout after which we regard a packet as lost. A better design choice would have been to count all the packets out, and accumulate all the replies asynchronously, matching them to the requests. This would allow potentially higher packet rates under Unix.

Example output from this program is show in Table 8: [23]

23. The minimum RTT of 0 is spurious and due to the poor clock granularity. Clock granularity of 10 msecs on current workstations is not unusual, and causes problems for very high speed analysis.

```
----------------------- 128.16.6.4 ICMP Statistics ------------------
pkt   pkts  pkts  pkts  pkt       rtt (ms)           tx t       rx t
size  /sec  tx'd  rx'd  loss   min   avg   max     (msecs)    (msecs)
64    3     7     7     0      0     20    70      448        448
```

TABLE 8. ICMP Statistics

Typically, an experiment would be to characterise some well known path through a section of the Internet (a difficult thing to establish in general) in terms of packet loss against packet size and offered packet rate, and delay distributions against packet size and packet rate.

Since ICMP/IP has no flow control or retransmission functionality, we can often extract a fairly accurate picture of the networks behaviour. In the case where outbound traffic interferes with reverse path traffic (e.g. under heavy loading), we may see some anomalous delays. Another shortcoming of this approach is that the request and reply packets are the same size, which does not reflect typical transport protocol behaviour or usage. In the latter case, typically heavy load is caused by bulk transfer, where large packets go in one direction, and small acknowledgements in the other.

5.3 TCP traffic generation

Again due to Muus and Slattery, we obtained a simple TCP traffic generator that runs on a variety of BSD Unix machines. We modified this program again to make paced user requests.

Typical parameters of interest are:

- The size of user write requests (i.e. where TCP does its pushes).
- The rate at which requests are offered to TCP.

A simple experiment to try with such a tool is to find the maximum throughput that a TCP can achieve under maximum load. Since TCP is a reliable and *timely* protocol, this should happen at the maximum offered user requests, but may not in some implementations.[24]

Typical output is shown in Table 9:

```
Tx side
ttcp-t: 0.0user 0.9sys 0:01real 84% 2i+2d 4maxrss 0+0pf 9+24csw
ttcp-t: 201032 bytes in 1.000000 CPU seconds = 196.320313 KB/cpu sec
ttcp-t: 201032 bytes in 1.180000 real seconds = 166.373146 KB/sec

Rx side
ttcp-r: 0.0user 0.3sys 0:01real 31% 2i+2d 4maxrss 0+0pf 140+138csw
ttcp-r: 201032 bytes in 0.380000 CPU seconds = 516.632401 KB/cpu sec
ttcp-r: 201032 bytes in 1.200001 real seconds = 163.600124 KB/sec
```

TABLE 9. TCP Traffic tool Example Measurement Output

Most of these numbers are to do with system overheads, not TCP or IP. The bytes/sec times are the only really useful numbers in determining TCP behaviour, except as regards the cost to the CPU.

5.4 Intrusive TCP Monitoring and Control

As described in the introduction, we made a number of simple changes to the 4.2BSD Unix TCP implementation to allow monitoring of connections.

---

24. For instance, the receive window offered is sometimes not adequate to permit the sender to fill the pipe, despite there being adequate buffering.

It is difficult to event drive a user program. Instead, we decided to record a window of events in a record associated with a monitored TCP connection (actually chained off the appropriate TCPCB). Secondly, we needed to allow a monitoring user program access to this control information (the TCPCB and the event record). This is done by adding I/O Control (*ioctl*) calls to Unix, together with a handler for this in TCP. The user program obtains a TCP socket (but does not necessarily use it). They then use the ioctls to access either the TCPCB associated with this socket, or another. There are two ioctls:

- To get a handle on the TCPCB appropriate to a given connection, we issue a *tcp_req* giving end point identifiers. Since TCPCB are chained together and uniquely identified by the source/destination host/port pair, it is easy to find the appropriate one.
- To get or set fields in the TCPCB and associated event record, we issue an ioctl for *tcpinfo*.

A number of the actual TCP modules are also altered to record certain information:

- tcp_input.c:

  record arrival of packet with type and time (e.g. sequence number plus has data, or pure ack).

- tcp_input.c:

  record transmission of packet with type and time (e.g. sequence number plus has data, or pure ack).

- tcp_timer.c:

  record types of timer that expire and associated sequence number of any packet rtx'd. We also added a function to allow pacing of TCP to IP transmission requests (along the lines of the process of dawdling for more user data which is used by some TCPs to optimise telnet packet usage).

- tcp_usrreq.c:

  Add alternative to cope with the request 'PRU_CONTROL'. In the new 4.3BSD code this alternative is trapped before the main switch to test which request is entered. This trap must be compiled out if monitoring is required. The contents of the trap must be included in the new test introduced later on to ensure that the same effect is produced if none of our ioctls have been called. A switch option for 'PRU_CONTROL' is added, which itself consists of a switch on the name of the request. The default case (i.e. none of the newly defined ioctls has been called must do the same as the original code. Finally two new routines need to be added, 'req_process' and 'tcp_fill_info'. The first handles ioctl requests that refer to a different tcp to the current one, and the second simply fills in the 'tcp_info' struct with the necessary data.

- In the file 'tcp_usrreq.c' there is a huge switch/case statement to process all kinds of things. One of the branches deals with 'PRU_CONTROL' which means ioctl requests. This in turn is a case statement, with simple ioctls being done on the spot (i.e. any that refer to the same socket as the request was issued on). The others are grouped together with a section of code that hunts for the TCP socket referred to in the request, and then calls the function 'req_process' to do the work. This function has a switch with a branch for each request. Add your request here if it needs to be able to operate on other process's TCPs, not forgetting to add it to the case statement which calls this function, where the other ones are.

We puzzled for some time about how we might add a facility for altering dynamically the round trip time estimator. It is tricky since Unix does not make sharing code between the kernel and a user program very easy. We decided that, although we could encode a number of simple RTT estimators in some symbolic way, the code to execute these within TCP was too difficult and dangerous to place in the kernel, hence we opted for rebuilding the kernel for each different TCP RTT estimator.

A suggestion for two different schemes was made by Craig Partridge:

1. "Assign a numeric code to each estimation algorithm. You would then read a variable which told you which algorithm was in use (for example, the value 2 might be for Mill's non-linear

filter combined with Karn's algorithm to select good samples). From this you would know that you could query for the values of alpha1, alpha2 and beta."

The problem with this approach is it requires a number for every algorithm, and code for all of the algorithms in the kernel.

2. Define a syntax for specifying the algorithm as a text string, and then support the ability to query for "alpha1", "alpha2" and "beta". For an example, see Figure 42:

```
RTO = (beta * Srtt);
if (S[i] > Srtt)
   Srtt = (alpha1 * S[i]) + (1 - alpha1) * Srtt
else
   Srtt = (alpha2 * S[i]) + (1 - alpha2) * Srtt

Uses Karn's algorithm to eliminate bad S[i].
```

Figure 42. RTT Smoothing Algorithm Specification

This is more elegant, but requires an interpreter in the kernel for this string.

In general, Partridge suggests that "the algorithms should be expressed as a sequence of equations that generate a result RTO (as above), and that all constants start with a lower case (and can be queried for) and all variables start with upper case. Finally define S[i], to be a well-known variable representing the most recent round-trip time sample taken (S[i-1] would be the sample before the most recent, etc)."

Having added these facilities, we can build monitoring programs to look at any TCP state/event sequences as we wish. Typically, we might want to track the behaviour of the *round trip time estimator*, or the behaviour of the transmit window, against time, or against packet transmission or reception.[25] This runs on local machine with altered kernel, takes as input a TCP connection number, and monitors that connection. It produces a stream of readings on stdout, formatted suitably for plotter. Eventually options could be provided to control what is monitored, and to set various controls in the connection. At present it extracts: bytes acknowledged; smooth rtt; idle time; no. of retransmissions; send window size; max segment size. (As the monitor has to be run on the same machine as the traffic generator program, it tends to impair performance of TCP. This can be reduced by running the monitor at a lower priority than traffic).

Unfortunately, simply copying a value into the TCP control block is not usually sufficient. In many cases these parameters are dynamically altered by TCP. When the experimenter sets them, he/she will want to be guaranteed that they remain fixed. A new set of flags can therefore be introduced in the control block, to define which parameters have been fixed, and tests inserted wherever TCP might want to alter such a parameter, to make sure it is allowed to. A further difficulty arises because not all the parameters potentially needing readjustment are held in the control block, if they were not subject to alteration in the lifetime of a connection before.

There are two types of such parameters. The first type is where previously the parameter was simply defined by a macro. A new field needs to be added to the control block, for which the default action is to initialise it from the macro at the start of the connection. Care must be taken to track down all uses of the macro and replace with the parameter. Examples are:

- Smooth round trip time calculation constants;

---

25. The intrusive measurement software was largely implemented by Steve Easterbrook while working at UCL under the supervision of the author.

- Timeout constants.

The second type is where such a parameter was not conceived of at all in TCP. An example is rate control. It would be desirable to control the rate at which TCP fires off packets, to study its effect. A simple method of rate control is that proposed for the NETBLT protocol,[Cla85a] which uses the concept of bursts of packets, where the burst size and burst rate are controllable parameters. To implement this, heavy-handed alteration of the TCP is required, and it is suggested that such alterations are surrounded by a different compiler switch so that they can be unilaterally excluded. Tools which attempt to use such facilities should check that the required **ioctl** s are implemented on the local machine.[26]

5.5 Passive TCP Monitoring

A different approach to monitoring TCP was provided by Van Jacobson of Lawrence Livermore Laboratories.[Jac90b] The problem with the method we used for monitoring each end of a TCP connection is that it monitored just that - each end. This meant that our observations interfered with the processes we were observing, and when we are interested in time related control, changing the performance of the system can be a problem. For the environment of interest (host to Ethernet to 64 Kbps SATNET to Ethernet to host and back), this interference was not really a problem, but in more high speed networks it would be.

The passive approach to monitoring TCP is based on the fact that the end hosts in an experiment are on Ethernets, and that it is relatively simple to monitor traffic on an Ethernet from a promiscuous host. The *tcpdump* program does just this. By using the Network Interface Tap mechanism in the Sun version of Unix, it is possible to access Ethernet frames and then filter out the ones relevant to a particular TCP connection (or any other protocol for that matter). The filtering is done on a simple pattern match on the headers (typically on protocol field and source and destination addresses/ports). As the monitoring host is simply spying on these frames, it has no effect on the connection being monitored. For each frame captured the protocol header information together with a timestamp are printed.

The main drawbacks to this tool are that the granularity of most Unix machine clocks is poor (typically plus or minus 10 msecs), and that the monitoring host may not be able to capture every frame if the monitored connection is between two very fast end hosts. However, many machines now are capable of capturing all the packets on an Ethernet although not doing a great deal else at the same time.

Typical output from a trace of a TCP connection is shown in Figure 43:

---

26. Historical Note:

The original UCL experiments on SATNET were carried out by Peter Lloyd and Jon Crowcroft. The system we had to instrument was a macro-11 version of TCP running in real time operating system, VMOS on a DEC LSI-11/23.

This system was instrumented in much the same way as above, except that the operating system could schedule events properly, and that we had direct access to a 1 microsecond line clock which allowed far more accurate timestamping. The summarised events were held in large buffers and written to a remote disk at the end of an experiment. These were generally sequence numbers of packets and acknowledgements and window sizes against time, or window size against packet sequence.

```
15:26:04.08  ego.1062 > SATNET-echo.2000: . 0:246(246) ack 1 win 16384
15:26:05.28  SATNET-echo.arpa.1062 > ego.2000: . ack 247 win 16384
15:26:06.10  ego.1062 > SATNET-echo.2000: . 246:492(246) ack 1 win 16384
15:26:06.10  ego.1062 > SATNET-echo.2000: . 492:738(246) ack 1 win 16384
15:26:07.48  SATNET-echo.arpa.1062 > ego.2000: . ack 493 win 16384
15:26:07.68  SATNET-echo.arpa.1062 > ego.2000: . ack 739 win 16384
15:26:08.52  ego.1062 > SATNET-echo.2000: . 738:984(246) ack 1 win 16384
15:26:08.54  ego.1062 > SATNET-echo.2000: . 984:1230(246) ack 1 win 16384
15:26:08.76  ego.1062 > SATNET-echo.2000: . 1230:1476(246) ack 1 win 16384
15:26:08.76  ego.1062 > SATNET-echo.2000: . 1476:1722(246) ack 1 win 16384
15:26:09.88  SATNET-echo.arpa.1062 > ego.2000: . ack 985 win 16384
15:26:10.06  SATNET-echo.arpa.1062 > ego.2000: . ack 1723 win 16384
```

Figure 43. Typical TCP Trace Output

This is from a connection between ego (a Sun 3/50 running Sunos 3.4, with Van Jacobson's modified 4.3 BSD TCP) and the SATNET IP echo site. The TCP MSS is 246 bytes, and the maximum window 16Kbytes.

The first field is the timestamp, then the source and destination addresses/hostnames. Then the most interesting fields are the start and end sequence numbers of the packet, the acknowledgement sequence number and the advertised window. Analysing output like this is discussed elsewhere (e.g. ACM SIGCOMM 88 proceedings for papers by Seo et al, and Van Jacobson).[27]

5.6 Setting up TCP experiments

First choose an interesting part of the Internet to look at (this is still not hard). Then go through the following steps.

1. Synchronise time between a source, sink and monitoring machine as well as possible (possible using a phone and stopwatch, or else rdate, or NTP if you have it).
2. Log into the remote machine and run your sink program and monitor the TCP socket it is using.
3. Log into an intermediate machine either on the same Ether as the source or destination machine, or if you are lucky, on an intermediate Ethernet and run your tcp trace program (e.g. etherfind, tcpdump etc), saving output to file.
4. Log into the local machine with the pre-prepared kernel and run your source program and monitor its TCP socket.
5. Create a window (preferably a "gfxtool" window) on the local machine, and in it run "monitor" with its output piped into the "plotter" program (see below). It will require the connection number given by "traffic" as input, (but in true user-friendly style, will not tell you that that is what it is waiting for), and then it will start plotting. And that's all there is to it.
6. Alternatively the output from "monitor" can be sent to a file, and then examined at a later date using "plotter <filename".
7. If you really want to get clever the whole lot can be pipelined: do (1) above and then try "traffic |monitor |plotter" in a suitable window. (If you want to plot the output on a different machine from the one with the hacked kernel because someone's been hogging it all day, you can do:

   ```
   rsh ego traffic hostname "|" monitor | plotter
   ```

   This runs "traffic" and "monitor" on ego and plotter in the window on the machine you're on.)

27. This output shows the classic binary exponential part of the "slow start" window opening algorithm.

Then collect together the source dump, the destination dump and the intermediate trace and look at them for a few days.

5.7 Analysing the TCP Experimental Output

5.7.1 Real time graphics

The program that monitored a TCP connection would generally be used just to dump sets of results to a file. As an educational alternative, a tool was developed to plot the output from the TCP monitor in 'real time'. The plotter takes a stream of data as input (either from file or pipe), and plots it against a time axis. Vertical scales are worked out dynamically by adjusting them during the first full screen of graph. Sample output from monitor (input to plotter) is shown below in Figure 44:

```
(time)#ack'd#sent#received#rtx timer#srtt#idle#rtxs#tx wdw#rx wdw#mss#
552234923 610000 3000 2000 0 2 0 0 0 2096 4096 1024
552234923 630000 2000 2048 0 2 0 0 0 2096 4096 1024
552234923 650000 2048 1024 0 2 0 0 0 1160 4096 1024
552234923 670001 1024 2024 0 2 0 0 0 2024 4096 1024
552234923 690004 0 0 0 2 0 0 0 2024 4096 1024
552234923 730001 0 0 0 1 0 1 0 2024 4096 1024
```

Figure 44. Sample TCP Monitoring Output

A more modest approach was to plot a single cartesian graph, given a stream of coordinate pairs. The first line should contain two hash (#) terminated strings which are taken as labels for the x- and y-axis respectively. The flag -b can be used to plot a bar graph instead of a line graph. The flag -d can be used to indicate a frequency distribution graph is needed in which case the input should consist of a single set of numbers, one per line. The frequency of each will be plotted. (Or if you want to get really sophisticated, each line can have a time (in seconds and microseconds), and a reading, in which case the distribution will be for the length of time spent at each value.

5.7.2 Statistical Analysis and Graphs

The most powerful (and surprising simple) way of analysing the performance of a TCP connection was again provided by Van Jacobson and is sometimes referred to as "Van Jacobson diagrams" (or Van diagrams, possibly not to be confused with Venn diagrams).

The trace from a TCP connection is hard to understand. Van Jacobson observed that simplifying the data to a plot of segment start send sequence against time provides us with a rather simple graph with a lot of information. This is exemplified in Figure 45:

Figure 45. TCP Behaviour with misbehaving Window

We see the same kind of trace in Figure 46, after treatment for misbehaving retransmission timer and errant window.

Figure 46. TCP Behaviour with Slow Start and Congestion Avoidance

A number of other ways of simplifying the data allow ack-number against time and number of retransmissions against time and so forth to be analysed and greatly ease demonstration of the protocol dynamics.

The packet send sequence number against time plot shows a number of pieces of information at once: The gradient at a point is the throughput; If the gradient goes negative, we have retransmissions. An increase in gradient is an opening window while a decrease is a closing window. A flat line is an idle TCP whereas a vertical line is a TCP that is dumping its window (e.g. after timeout and retransmission).

5.8 Measurements on SATNET

This section is about measurements analysis and performance tuning of the Transmission Control Protocol (TCP) and the Internet Datagram Protocol (IP) in the network environment found between the UK and the US.[Seoa]

The route we were interested in included the Atlantic packet satellite, and parts of the US Internet (including the ARPANET). This contained such a diverse collection of hops, gateways and switches that formal analysis of the system is extremely difficult.

5.9 Background and Motivation

The Atlantic Packet Satellite Network (SATNET) was used for some years as a main route between UK MoD research networks at the then RSRE (now called DRA Malvern), NTA and NDRE in Norway, CNUCE in Italy, FGAN in Germany and STC in Holland, and the US for interactive traffic.

For some time, the perceived performance of applications across these paths was very much lower than we expected for a system built of links around 56Kbps (the UK end was 64Kbps, while the SATNET itself is 2*64Kbps, but the US end was 56Kbps). Gateways that should normally have been able to drive such lines adequately fast.

The user generally perceived data rates around 3Kbps or even lower, and frequently connections would be lost indicating incredibly high packet loss or routing problems. Since the actual bit error rates on the various hops are not particularly high, we suspected protocol design or implementation problems as a prime cause.

The SATNET task force divided the task of identifying the problems and curing them into 3 parts.

1. Identify each hop's network behaviour. By hop, we mean SIMP-to-SIMP, Gateway-to-SIMP, Host-to-Ethernet-to-Gateway, etc.
2. Characterise ip behaviour.
3. Find any problems with TCP.

5.10 Choice of Transport Protocol to Measure

We chose to measure TCP performance rather than any other transport protocol because its the internet standard transport protocol for most applications. It has adaptive timeouts and adaptive windows, and therefore has complex behaviour when the underlying network quality of service varies.

5.11 Characterising Performance

There are two important and different views of protocol performance. On the one hand, the end user is interested in high throughput for bulk data transfer, and low round trip delays for interactive use. SATNET has reasonable underlying throughput, but a high basic delay. On the other hand, the network provider is interested in high utilisation of the net, together with fair sharing of network resources. SATNET is intrinsically a shared resource, being a broadcast based medium.

All studies of TCP behaviour therefore concentrate on methods of maximising throughput and minimising delays, while minimising unnecessary retransmissions. Some more recent studies have looked at fair sharing. In measuring the performance of a given TCP implementation, we need to examine all of these factors.

5.12 What Parameters and Algorithms?

A given TCP implementation can be correct, and yet will perform differently to another. When measuring a TCP, we need to systematically characterise the algorithms and parameters used for a variety of parts of the protocol so that comparison with other implementations or parameters is possible. We have excluded certain parameters as not seeming to be relevant to perceived problems over the SATNET.

We informally surveyed a number of experts' opinions on which parameters would be most critical in the performance of TCP, and accumulated the following list:

- Packet Sizes

  TCP negotiates a Maximum Segment Size (MSS) on opening a connection. This is currently a simplistic switch between local net MSS and anything else. The MSS for SATNET did not need necessarily to be the same as the SATNET maximum IP size (256 bytes MTU - 20 bytes IP header

- 20 bytes TCP header - 6 bytes additional overhead = 210 bytes user data), since there was a reduction in header overhead by sending larger IP packets. The tradeoff is against the loss of more complete IP packets as we hit the error rate. Theoretically it is trivial to find the optimum (for a bit error rate, and given header overhead, optimum pkt size calculation...). However, the effect of fragmentation is not necessarily so simple, in that intermediate devices on a route may treat fragments differently (for instance by forwarding them back to back).

We measured the performance of bulk transfers for a variety of MSS values to see if the optimum was as IP level measurements and theory would predict.

- Timers

Fixed timers failed when trying to run over SATNET, and so RSRE introduced the simplest possible feedback mechanism for adaptive retransmission timers, using linear feedback from the mean measured round trip delay.

This mechanism fails to take account of the errors in the estimation method or the underlying technological mean delay, and so several papers in the literature suggested that the mean, together with the variance on round trip time, should be used as a better estimator.

We compared the accuracy of the "mean+smoothed mean difference" algorithm, with the original RSRE algorithm.

- Windows

Many early TCP implementations use the receivers advertised window, with the (often inaccurately) estimated round trip time to fill a pipe between the transmitter and the end of a receivers buffers. The problem here is that the receiver closing the window is certainly a flow control mechanism. However, the receiver cannot tell the transmitter anything about network conditions, and so cannot use the same mechanism for congestion control. Loss of acknowledgements is used to back the transmitter off. However, many TCPs simply compound network congestion by retransmitting the entire window repeating an action that was a probable cause of loss in the first place.

Recent developments suggest a "slow start" algorithm to opening the window after loss, gathering speed with each acknowledgement until some threshold level of the previous problem window size. Then a linear increase (to try and find the optimum "congestion window" without causing the window size to oscillate around the right value). The exponential to linear switch is designed to be stable where the network is quick to fail and slow to recover.

A recent refinement to slow start can be found in Wang.**Wan91a**

We compared the throughput and number of unnecessary retransmissions for a conventional windowing TCP, with a TCP using the slow start and congestion controlled window mechanisms.

5.13 Naive Throughput and Delay Calculations

Figure 47 shows a naive calculation of throughput and delay expectations for a UK site to SATNET to another site.

- Assume a packet size of 256 bytes, with 210 bytes of user data.
- Assume the terrestrial tail is 56Kbps.
- Assume the basic delay over SATNET is .36 seconds from Earth station to Satellite.

```
        56Kbps = 7192 bytes/second
        End-to-End delay: .72 seconds

=>        pipe size = .72 * 7192 = 5178 bytes
=>        5178/256 pkts/second rounded down
=>        20 packets per second
=>        20*210 = 4200 user data bytes

or        33600 bps of user data.
```

Figure 47. Theoretical SATNET Throughput

Firstly, this will only occur if there are no queuing delays in the gateways or earthstation equipment. Secondly, if the window offered by the receiver and packet rate/transmit window used by the transmitter are at least 5178 bytes.

- Assuming Poisson arrivals at SIMP, and negligible queues in butterfly (unreasonable, since this assumes packet arrival times are independent).
- Assuming exponential service times in SIMP.

Figure 48 shows the calculation of the SIMP queues. Table 10 shows a range of results for both queue sizes and resulting waiting times.

```
    mean queue size = rho/(1 - rho)
    mean waiting time = Ts * rho / (1 - rho)

where
    rho = lambda*Ts

and
    lambda = the mean message arrival rate,
    Ts     = mean message service time in SIMP.
```

Figure 48. SIMP Queue Length and Service Time Analysis

With uplink speed 64Kbps per channel and packet size 256 bytes on the air, Ts = 256*8/64*1024 = 30 ms (ignoring processing time!!). So we can plot mean queue lengths and waiting times against packet arrival rates (e.g. offered load dictated by receivers offered window).

packet rate	window	queue size	waiting time
1	151	0.03	0.00
2	302	0.07	0.00
3	454	0.10	0.00
4	605	0.14	0.00
5	756	0.19	0.01
6	907	0.23	0.01
7	1058	0.28	0.01
8	1210	0.33	0.01
9	1361	0.39	0.01
10	1512	0.45	0.01
11	1663	0.52	0.02
12	1814	0.60	0.02
13	1966	0.68	0.02
14	2117	0.78	0.02
15	2268	0.88	0.03
16	2419	1.00	0.03
17	2570	1.13	0.04
18	2722	1.29	0.04
19	2873	1.46	0.05
20	3024	1.67	0.05

TABLE 10. SATNET SIMP Queue Length and Wait Times

In fact these queue sizes were exceeded immensely since the SIMP processing power was such that there was a cpu queue as well as an outgoing link queue. So we note that the theoretical maximum throughput without problems is 33600 bps. The theoretical minimum delay is 30 ms (1/32 second).

5.14 Maximum Segment Size Choice

There is a tradeoff between errors and header overhead which defines how we choose a TCP MSS. The SATNET had a packet size of 210 bytes, into which the IP packet had to fit. The IP header overhead was 20 bytes, while the TCP overhead was 20 bytes. A single TCP segment has a single 20 byte TCP header. If the TCP segment was larger than the maximum IP fragment size, we get two (or more) IP packets, but still only one TCP header. As the TCP segment size increased, the number of IP fragments increased, but the chances of any one fragment being in error increased. If any single IP fragment was in error, the entire TCP segment was lost (after the reassembly timeout in the IP receiver). This caused retransmission of the entire TCP segment, and therefore all the IP fragments.

The 4.3BSD TCP has two choices for IP size, and one for TCP MSS. There has been discussion (on the TCP-IP mailing list and in the End-to-End task force, and there is now an RFC (Fragmentation Considered Harmful)), which airs many of these issues. Basically, a mechanism for IP optimum fragment size and TCP MSS size discovery would be useful [and several mechanisms have been suggested]. For the purposes of these experiments, however, we chose to configure a TCP with 2 fragment sizes, one for LAN or directly attached subnets, and the other for off-site connections.

We can form a simple approximation for an equation for the expected number of packets on the network against TCP segment size easily. Assuming the packet error rate << 1, this is as shown in Table 11. Some resulting values are shown in Table 12.

```
meanpktssent = (segmentsize/fragmentsize)
               ------------------------
               (1 - packeterrorrate * (segmentsize/fragmentsize))

             = 1 / (fragmentsize/segmentsize - packeterrorrate)
```

TABLE 11. Theoretical Packets Sent versus Segement Size

So we can tabulate mean packets sent against segmentsize for a given error rate (making things easy, we choose to guesstimate SATNET errors at 1% per SATNET packet size...).

segment size	mean packets sent	bandwidth utilisation
256	1.0101	0.8121
512	2.0408	0.8422
768	3.0928	0.8462
1024	4.1667	0.8438
1280	5.2632	0.8387
1536	6.3830	0.8323
1792	7.5269	0.8252

TABLE 12. Theoretical Packets Sent versus Segement Size

We knew that the SATNET probably had the highest actual bit error rate of the hops we were measuring, what it was (10^-6), and also that the SATNET had the smallest packet size, hence we calculated that a 768 byte IP fragment should be optimal as far as errors were concerned. We only tried 256 and 512 bytes since some fragmentation effects are undesirable. However, as mentioned above, there are other reasons for not fragmenting IP, such as the lack of flow control between IP fragments causing spurious effects in some queues and upsetting some of the mechanisms now in TCP (see below).

5.15 Retransmit Timers and Round Trip Time Estimators

The conventional algorithm (RFC793 "RSRE") **Pos80a** is a simple estimator for the round trip time is based on measurement of the time between sending a TCP segment and receiving the acknowledgement for sequence numbers in that segment.

There is a classic tradeoff between prompt retransmission (faster throughput for a single user) and delayed retransmission (less wasted bandwidth on the channel for other users. In fact, unnecessary or over-keen retransmission can cause problems even for a single user when there is a bottleneck queue in the system, recovery from over-running the queue is usually slower than causing over-run, thus reducing bandwidth further than expected.

Several things can go wrong with this method of estimation:

- There may be bugs in the implementation - for instance timing a retransmitted segment can distort the estimate since an acknowledgement may be for the original transmission etc.
- The feedback into the estimate has a fixed contribution from past measurements and the latest measurement. Round trip times may change smoothly when traffic is changing smoothly, but abruptly when routes change or traffic alters suddenly.
- There is a fixed estimate of error in a round trip time estimation. This is not often realistic since we trust highly variable measurements less than less variable ones (and wish to play safe with retransmissions).

The contribution of these factors and more may cause a TCP connection never to reach a stable/correct estimate of the round trip time, and therefore to vary the total rate at which it transmits segments (as new segments and retransmissions get sent). This can cause oscillation of the queues in switches/routers which simply exacerbates the situation.

5.16 Windows, Retransmission and Congestion

The standard TCP adaptive sliding window is very often not implemented fully. Also, it is often not based on the correct data. The purpose of the window is twofold:

- For the receiver to flow control the transmitter (advertised receive window).
- For the transmitter to gain better throughput by filling the network *pipe*.

The first of these is usually implemented correctly. However, finding the size of the network *pipe* is not trivial. One method might be to choose the estimated round trip time of the channel, and known lowest bandwidth hop (perhaps kept in routing tables). This makes no allowance for other users, and has another serious problem: With the standard TCP protocol, it is impossible to tell the difference between loss of packets due to errors and congestion. However, the path under consideration was somewhat complex compared with other parts of the Internet and for the following reasons, it is actually very hard to distinguish between overload, congestion and genuine loss: The European users of SATNET generally accessed hosts past the end of SATNET somewhere in the US Internet; the US Internet was extremely congested during the period that these experiments were being done. the SATNET had a significant error rate compared with many other paths (typically 1%); some of the switching components on the SATNET (for instance the SIMPs) could not handle packets of some sizes at the same rate as the incoming links.

The analysis here is partly due to Jacobson: [Private Communication to the End-to-End and SATNET Task Forces].**Jac88a**

First, we define the following terms:

- D bytes of data to send
- window size of W
- round trip time of T.
- A fraction L of the packets sent are lost.

Note that the operating regime for this window scheme is one where you cannot get information back from a receiver before a window W is sent. To puit this another way, the window is always smaller or equal to the pipe size.

We have to send every packet at least once so the minimum transfer time is DT/W. Accounting for retransmits, retransmits of retransmits, etc., the total number of retransmitted packets is D L/(1-L). In a standard-conforming RDP,**Vel84a** it takes 2T to find out that you need to retransmit plus another T to get the ack. So the retransmits add a time D L/(1-L) 3T/W to the transfer time. The total time is then DT/W (1 + 3L/(1-L)). I.e., the effect of loss is to increase the minimum transfer time by a fraction 3L/(1-L). Maximizing throughput means minimizing this fraction. To find the window size that maximizes throughput, take the partial with respect to W, set it to 0 and solve for W. (note that minimizing the number of packets sent corresponds to minimizing L/(1-L). Thus the W that maximizes throughput also minimizes total traffic.)

Now lets look at the network.

- N simultaneous users on the average
- all use the same window size
- ALL see the same round trip time.

With no loss, throughput is N W/T. Throughput after loss is at most N W/3T. Since only an in-sequence ack opens the RDP window, no-loss and loss are mutually exclusive and bandwidth needed for user data will be the weighted average. I.e., L NW/3T + (1-L) NW/T or NW/T (1 - 2L/3). Add the L NW/T we lost when we tossed the original packets and we end up with a total bandwidth needed of NW/T (1 + L/3). So, for a network with fixed capacity running at capacity, the effect of loss is to reduce the available bandwidth of the by a fraction L/3. Maximizing utilization corresponds to minimizing this fraction.

These problems led to four major modifications to the TCP we were measuring (all modifications due to Van Jacobson):

- Dallying of acknowledgement removed (waiting before acknowledgement in case more data is received and can be acked degrades performance of self clocking).
- Phil Karn's clamped mean plus mean difference retransmit algorithm added.
- To prevent the TCP *dumping* its entire window worth of data into the network when starting a connection or when retransmitting after a packet loss, the *slow start* algorithm was added. This entails using acknowledgements to clock out new transmissions.
- A congestion avoidance scheme was added to the BSD TCP to allow it to find the *pipe-size* in the presence of congestion. This is the exponential to linear threshold for opening a window after loss, set at half the last successful window (i.e. half the window size at which we had a loss).

  This is where we lose badly on a channel with real loss as opposed to congestive loss, since every real loss causes this algorithm to fire (but there is no choice since we cannot tell real loss from congestion - see the discussion at the end of the chapter).

At the same time there was some discussion of the use of Source Quench messages from gateways to introduce delays between packets at the transmitting host. We modified the IP in our test system but since the bottleneck was the SIMP rather than the gateway, we did not receive source quench messages. One experiment was tried tentatively using TCP segment loss to introduce IP packet delays, but we detected no significant effects. (One problem was the granularity of the clock which did not allow fine enough tuning of the interpacket delays).

5.17 Traffic generators

First of all we need a way to generate controlled traffic. Since we were interested in bulk transfer performance, which would stress the SATNET as much as possible, we used a fairly simple program which attempts to drive random data from memory as quickly as possible over a TCP connection. The source and sink machines used were originally LSIs, but in the latter experiments, high speed workstations capable of Megabit data rates were used. This meant that benchmarking the traffic generated when driving over a 64Kbps line was fairly accurate. Typical transfers of 1 Megabyte were used to gain sufficient data to eliminate anomalies and detect any regular behaviour.

5.18 Intrusive Instrumentation

Initially, UCL developed an instrumented version of TCP which allowed a traffic generator to set parameters such as MSS, maximum window sizes, weights in the sRTT calculation and so on, for TCP. This ran on LSI's under a local real time operating system. The traffic generator was integrated with the TCP. Based on this experience, we instrumented the 4.2BSD TCP by providing a control path for a user to access the entire Protocol Control Block for any given connection, and access it per network or timer event. This proved a useful educational tool, since we could graph the behaviour of a connection as it ran, in real time.

5.19 External Observation

Van Jacobsen provided the task force with an invaluable tool for observing a TCP's performance without altering the actual implementation. This program runs on a different machine to the source or sink, but on the same LAN as at least one end of the connection. The program simply filters all the packets for a given connection for later analysis. The packet rates that can run over SATNET are easily coped with by this system.

A collection of post processing filters were also provided, which allow close analysis of the timing of packet transmissions, acknowledgements, retransmissions and so on. For instance, the most useful way of understanding a TCP behaviour has been to graph sequence number versus time for Packets and Acknowledgements at each end of a connection.

5.20 The Measurement Environment

The most common way that hosts have been recently connected to the internet is via a gateway from a LAN. UCL, CNUCE and NTA all had such a configuration, and all have Unix based workstations from which we could generate traffic and for which the same TCP could be provided. The characteristics of the LANs (whether Ethernet, Token Ring or Slotted Ring) are such that any errors, congestion or delays would be completely negligible compared with those of the SATNET portion of an experiment path. This topology is shown in Figure 49.

Figure 49. TCP Measurement Environment Topology

At first we frequently made use of the IP echo hosts in the SIMPs, but then changed to running the sinks in hosts at other sites as there was detectable interference between outbound and return traffic under high load due to the limits of processing power in the SIMPs.

For a range of MSS, maximum windows, timer and window algorithms, the paths measured were:

1. From a TCP source at UCL, via the IP echo site at the nearest SIMP to a TCP sink at UCL

   0 -> 1 -> 0

2. The same via the logical SATNET IP echo site.

   0 -> 1 -> 2 -> 1 -> 0

3. The same via the IP echo host in remote SIMP

   0 -> 1 -> 2 -> 3 -> 2 -> 1 -> 0

4. From a TCP source at UCL to a TCP sink at a remote site, for both CNUCE and NTA, and for the reverse path.

   0 -> 1 -> 2 -> 3 -> 4 -> 5

   and

5 -> 4 -> 3 -> 2 -> 1 -> 0

Some of these experiments were repeated for multiple simultaneous connections from TCP sources a UCL to sinks at one other remote site, and for differing remote sinks.

5.21 Discussion of RTTS and Windows

1. RTT Estimation

   These experiments were done between a Sun on the UCL LAN and the satellite echo host. We used XTCP, (an experimental TCP based on 4.3 BSD), with an MSS of 210 octets and maximum receive window of 16 Kbytes. There are two RTT vs. RTX graphs. The Time axis is half the measured or estimated round trip.

   In Figure 50 we show RTTs where the Smoothed Round-Trip Time (SRTT) algorithm of RSRE has been used, SRTT does not follow the measured RTT closely. It is higher than it should be due to the high basic delay over SATNET, which causes errors in estimation to be exaggerated. The addition of variance to the calculation of SRTT produced a marked improvement. The SRTT much more closely follows the RTT curve. The peak near the beginning of each sequence plot corresponds with the initial high load as TCP finds a correct window (see below). In the steady state, the mean and variance of the round trip time fit closely with those seen directly at the IP level.

Figure 50. Smoothed Round Trip Time Results

2. MSS Selection

   These experiments were done between a Sun on the UCL LAN and the satellite echo host. The XTCP was operated with a maximum receive window of 16 Kbytes and 2 different MSS's. In figure 51 we plot packet sequence numbers versus time for the two MSS experiments. The X-line is with an MSS of 210 octets, the O-line is with an MSS of 420 octets. [28]

28. Picture courtesy BBN

Figure 51. MSS Selection Results

It was hypothesised that for the larger MSS, there might be more retransmissions due to loss of IP fragments than there would be for the smaller MSS, but that there would be more header overhead for the smaller MSS. The effect on throughput of MSS (at least at one times and two times the SATNET IP MSS) does show in these tests though it is perhaps not as bad as halving the throughput. One effect seen here is the more catastrophic collapse of the connection soon after startup, for the case when the TCP_MSS exceeds the IP_MSS.

3. Retransmission

   These experiments were done between a Sun on the NTA-RE LAN and one on the UCL LAN, using XTCP with MSS = 420 octets, and a maximum receive window of 16 Kbytes. The first TCP graph shows retransmissions after loss/congestion for two retransmission strategies with packet sequence number plotted against time. 'Standard' TCP retransmitting whole window is indicated by dots, and $+$'s indicate 'Slow Start' TCP which opens up the window slowly after packet loss.

   Referring back to figure 45, we see that for 'Standard' TCP, after a packet loss there are a significant number of retransmissions. It also shows that for "Slow Start" TCP there are far fewer retransmissions as the window is opened first exponentially and then linearly. Basically only the packet that was lost gets retransmitted. In SATNET, packet loss was typically due to bit errors rather than congestion, so losing one packet did not imply the loss of any of the following packets. For packet loss due to congestion, the retransmissions incurred in the first case might not turn out to be as ''unnecessary''.

4. Adaptive Window Sizing for Pipe-size

   These experiments were done between a Sun on the NTA-RE LAN and one on the UCL LAN, using XTCP with MSS = 420 octets, and a maximum receive window of 16 Kbytes.

   The second TCP graph in figure 46 shows retransmissions after loss/congestion for two adaptive window sizing algorithms. Packet sequence number is plotted against time to show the differences between 'Standard' and 'Congestion Window' TCPs.

   The figure shows the effects of retransmitting the whole window after a start or packet loss. If there were congestion, this would only make matters worse. If there is random packet loss, as in SATNET, this causes unnecessary retransmission of packets that were not lost and lowers throughput. The second TCP figure shows that the use of a congestion window avoids the problem of ''unnecessary'' retransmissions. However, when the packet loss is due to random bit errors rather than congestion, there is unnecessary throttling of transmissions and therefore lower throughput. Each glitch in the line is a lost packet. Counting through these we can see around a 1 packet loss

rate. (Since TCP is using an MSS of two IP maximum fragments, this represents closer to 2 TCP packet loss). At this level, the exponential/linear congestion algorithm can never find the right bandwidth. Analysis shows that we can expect less than half the bandwidth that IP might achieve. These results point to the problem in interpreting packet loss as an indication of congestion in an environment where packet loss is due to bit errors.[29]

5. SATNET and ARPANET

   This experiment was a run of standard TCP over a path involving both SATNET and ARPANET, with an MSS of 512 octets, and an advertised receive window of 4 Kbytes. The path went from the UCL LAN to EDN-VAX.ARPA.

6. Multiple connections

   This experiment involved two TCP connections from a Sun on UCL LAN to a Sun on NTA-RE LAN. The TCP was XTCP with an MSS of 420 octets, and an advertised receive window of 8 Kbytes. The third TCP graph in Figure 52 shows how the congestion window algorithm shares bandwidth fairly among TCP connections.

---

29. On later analysis, it was pointed out that there is an anomaly in some of the test figures. However, this does not actually affect the conclusion that this congestion avoidance scheme can cause throughput loss (albeit at no cost to the network) in the presence of high loss due to genuine errors:

    The ttcp output in one test took 693 seconds for a throughput of 2.88 Kbps. The tcpdump output says the test took 770 sec for a throughput of 2.65 Kbps. The end-to-end task force agreed that there was evidence in the trace for the 2.65 Kbps rate. It is possible that the clock of one of the end systems was adjusted by some time synchronisation protocol, whilst the tcpdump monitoring machine shows a monotonically increasing clock in the trace, and is thus a more plausible source of data rate.

    With a user data size (mtu) of 420 bytes, there should be 488 bytes going through the SIMP for each data packet (2 frags, 1 w/20 byte tcp hdr, both w/20 byte ip hdr & 4 byte butterfly hdr = 420 + 20 + 2*(20 + 4) = 488). There's also an ack for every 2 data packets that contributes another 44 bytes (20 tcp, 20 ip, 4 butterfly). If we charge each data packet for 1/2 an ack, the SIMP sees 488 + 44/2 = 510 bytes for every 420 bytes of user data. Or, the total unidirectional throughput through the SIMP is 21% higher than the data throughput. 2.65Kbps * 8 = 21 Kbps * 1.21 = 26 Kbps. So, the SIMP should have been handling around 52 Kbps. An awk script to compute total bits sent to the SIMP vs. time shows a line whose slope turns out to be 26 Kbps with a couple of brief (˜30sec) excursions to 30 Kbps.

    This shows an interesting example of self-organizing behaviour: At around 20sec, the connection is in equilibrium. Because of slow-start, the acks and data are pretty well interleaved. But because of cookie crumb effects, the acks almost immediately start diffusing towards the "late" end of the rtt slot and, by 100sec, all the sends show up at the early end of the slot & all the acks at the late end (you can see this as a "staircase" effect in the plot that is barely noticeable at 20sec and grows to be substantial at 120sec).

    We can measure the average amount of the slot occupied by the acks (which seemed to be 1.6 sec) and the number of acks (which was 8 at equilibrium). We know that there's one ack generated for every 2 packets and the acks are generated at exactly the rate that data packets come out of the SIMP. Thus 16 packets times 488 bytes/packet times 8 bits/byte in 1.6 sec = 39,040 bps = 38 Kbps.

    This flow separation of the sends and acks (which showed up in earlier data) is the source of the 2.65 Kbps prediction. We say that at equilibrium the sends and the acks are going to occupy non-overlapping parts of an rtt slot of length R. Say that at equilibrium we generate n packets of total size P bits. If the channel bandwidth is B, the ack portion of the slot must have duration nP/B. Two packets will be sent on the receipt of each ack so, if the ack spacing is preserved as the acks flow back through the channel, the send slot will also have duration nP/B. However, because of the PODA piggybacking & the symmetry of an echo test, the acks get compressed together and come out the other end only one packet time apart rather than two Thus the duration of the send slot is half the ack slot. Thus we have:

    $R = 1/2\ nP/B + nP/B = 3/2\ nP/B$

    but the effective throughput for the test is the rtt divided by the amount of data, or nP/R. Substituting the above expression for the R, the nP's cancel and we end up saying the effective throughput for an echo test will be 2/3 B. That says that for a 38 Kbps channel you should have gotten 25.33 Kbps. A least- squares fit to the tcpdump data shows 26.43 Kbps.

    The CNUCE data could have been interesting (since it was not an echo test) but was limited by the receiver advertising only a 4KB window. The average send-to-ack time was 2.6 sec so the maximum possible throughput was 4/2.6 = 1.5 . Several short portions of the trace (short = ˜30 sec) seemed to go at this rate but, overall, it the dismal 1.1 KBps because of a very high error rate (2% or 97 packets resent for 4973 sent). With this high error rate, it was going to be hard to get good throughput even when the window was a reasonable size.

Figure 52. Two TCP Connections Sharing Bandwidth

5.22 Related Work

Related work on TCP/IP [Pos80a] performance specific to this chapter include Clark's initial work on windowing,[Cla82b] Cole and Lloyd's comparison between TCP and TP4 windowing and acknowledgement,[Col86a] Van Jacobson's seminal work on end-to-end congestion avoidance,[Jac88a] Nagle's early work on congestion control,[Nag84a] Lixia Zhang's work on TCP timer estimation,[Zha86a] Peter Lloyd's thesis work on early SATNET measurements,[Llo86a] and the earlier work by Edge on flow control and adaptive timeouts,[Edg83a, Edg84a] togther with Raj Jain's work on timeout schemes.[Jai86a]

5.23 Conclusions

Measurements of TCP including the RTT estimation and slow start and congestion avoidance algorithms show the importance of good engineering of communications systems. The mechanisms employed are both general, and conform to the end-to-end arguments in system design. The measurement paths are characterised by a large "pipesize" or bandwidth/delay product, and the findings are therefore relevant to future high speed terrestrial paths, such as those likely to be used in Broadband ISDN networks such as SuperJANET.

6. Chapter 6: Managing the Interconnection of LANs

This chapter compares the problems of interconnecting LANs running connectionless network protocols by link-level bridges, and by internetwork level relays. Issues addressed include access control, remote management and routing mechanisms. We introduce some algorithms that might be used to resolve addresses and routes in interconnected LANs. The material presented in this chapter is background to original material presented in the next chapter on access control for bridged LANs.

Chapters 3,4 and 5 described the design and performance of transport level protocols for distributed applications. The protocols described in chapters 6 and 7 are distributed in a different way: Routing and management protocols are embedded in a system to provide fault tolerance and efficiency (amongst other goals). Unlike applications which are written frequently and in large numbers, embedded distributed systems like this are implemented less frequently, but used most frequently. This leads to the use of techniques that are even more lightweight than some of those described in the introductory chapters 1 and 2, but much harder to reason about.

6.1 Introduction

Many popular Local Area Networks currently in use support connectionless mode network protocols. One widespread example is Ethernet running the Internet Protocol[Pos81a] (IP). Hosts implement an end-to-end protocol[Pos80a] (TCP) that performs all the necessary error recovery and flow control. A similar architecture can be constructed using the ISO connectionless mode network service, with the hosts implementing Class 4 of the ISO Transport Protocol.

Most LANs have a limited geographic extent, usually of order one kilometre. They also have a limited number of hosts that may be attached, typically of order one hundred. Many institutions have requirements that extend beyond these limits, although not far enough to justify building a WAN or paying for PDN use. The rest of this chapter is structured as follows:

First we discuss protocol dependencies of different interconnection[Cro88b, Cro88c] tehniques. Next we look at mechanisms for Address Resolution. Then we examine Faults Isolation, and how bridging and routing interact with it. Then we look at topological control: When forming topologies more complex than a bus or a ring, there is the opportunity for redundent paths and faulkt toloerance, but this has the consequent danger of creating loops. Subsequently, we examine Remote Management. The penultimate section looks at some performance gains and penalties of different interconnection techniques. Finally, we look at the common (but poorly catered) for case for remote LAN interconnection, over connection oriented public networks, such as ISDN.

6.2 Protocol Dependencies

Internet relays traditionally support the interconnection of heterogeneous networks. Examples of these are the Internet Gateway, PUP Gateways and Xerox Communication Server Routers.[Hin83a, Bog80a] The intention behind interconnection at this level is to isolate higher level protocols from the specifics of each link or subnet, and provide a homogeneous network service over a variety of hardware link and network levels.

By contrast, bridges connect networks which are identical or similar at the lowest level. In fact, the only requirement is that the networks, subnets or links have similar framing and addressing. However, we shall see later that presence of hardware level broadcast may radically affect the design of such devices. The bridge mechanism means that entirely different protocol architectures may coexist on a set of inter-connected LANs.

6.3 Address Resolution and Mapping

6.3.1 Internetwork Address Mapping

When LANs are interconnected by internet routers, there must be some mechanism for hosts to locate routers, and each other, since the internet routing layer will use some internet addressing mechanism. Which scheme is used will determine how much location information has to be held where. The spectrum stretches from full distribution of the data to the hosts, to centralised location of the data in routers and possibly nameservers.

This *address mapping* can be achieved in a variety of ways.

- Fixed mapping

  There may be some simple direct mechanism for systematically mapping between local hardware addresses and internet addresses. Such a scheme was used at UCL for running Internet Datagrams over Cambridge Ring frames, where the local host part of the IP address was the same as the basic block address.**Llo80a**

  This is mainly useful where all the hosts in the system only use internet type services. This is because a one to one address mapping between the internetwork layer and the local network layer implies that there is global agreement on high level protocols. Otherwise a host might be sent internetwork packets which would have no meaning to it.

- Centralised Control

  A Nameserver may be used to allocate dynamically and to hand out mappings. Hosts find the mapping from an internet address to a local network address by querying the nameserver.

  Such a role was fulfilled by the Universe nameserver, in collaboration with the ring-ring bridges.**Les81a, Ada83a**

  When a host on one ring wishes to communicate with a host on another, it first asks the nameserver where the second host is. The nameserver communicates with the first hop ring-ring bridge, establishes a port for forwarding traffic, and then returns this address to the first host.

  Typical design decisions are the include whether to distribute the mappings between multiple nameservers, or only hold relevant local net specific mappings in each nameserver. If the former choice is made, a further refinement is whether the lookup is recursive or iterative through the nameservers.

- Distributed Control.

  The Address Resolution Protocol is widely used on LANs with broadcast to fully distribute responsibility amongst hosts for providing internet to physical address mapping.**Plu82a** When a host A wishes to communicate with host B, and possesses its internetwork address, it broadcasts a request on the local network with B's internetwork address, asking for the local network address of B. B then replies, also noting A's internetwork and local network addresses in the request.

None of these mechanisms, except the Universe nameserver, was originally designed to function in a multiple LAN system.

There have been three suggested enhancements to address mapping that allow ARP type schemes to still be used, and are commonly implemented.

1. Flat Addressing

   With this scheme, the bridges forward address resolution requests made by hosts for hosts on other LANs, onto those LANs, making sure not to loop any such requests. This may result in large numbers of broadcasts where the number of interconnected LANs is large. The size of address caches in hosts may get correspondingly large.

   Another problem is that the router is now forwarding two types of protocol. This means that it loses one of its advantages in that the LANs either side must support both of these protocols.

2. Hierarchical addressing

   Here the internetwork layer has hierarchical addressing structure, where some part of an address indicates which LAN a host resides on. Then hosts may using an extension of the internetwork routing system to locate first hop LAN internetwork routers. The problem here is managing the allocation of address space to deal with interconnected LANs with differing numbers of hosts.

3. Proxy ARP

   This system involves a router acting on behalf of a host broadcasting a request for a host on another LAN. Here the router knows, either from hierarchical addressing or from learned tables, which collection of hosts are on which LAN. On seeing a request on one interface for a host on another, the router replies on behalf of the target host.

   Managing the tables in the router can be complex, and there are no mechanisms for coordinating proxy ARP table management with dynamic routing.

6.3.2 Addressing in bridged LANs

Because MAC level bridges operate transparently, there is no requirement for address resolution in hosts beyond that for a single LAN. One consequence of this for a large interconnection of LANs is that the number of ARP type broadcasts and the size of corresponding address caches in hosts can become large.

The bridges themselves have to accumulate tables of physical addresses of hosts, since there is no other automatic way of determining which hosts lie on which LANs. This process of learning the list of hosts per LAN becomes complex when the are multiple interconnected LANs and is discussed below.

6.4 Fault Isolation

Some types of LAN are prone to low level failures that have complex higher level consequences. Where the LAN technology does not have active components as part of its connectivity, this can make fault isolation extremely difficult. For example, a faulty transceiver can result in the unexpected low performance of a transport protocol. With large numbers of hosts on a single LAN, this kind of problem can be extremely hard to locate. By dividing the hosts amongst several LANs, we simplify the search.

6.4.1 Traffic Localisation

Aside from physical scaling problems, one of the main purposes behind using multiple LANs is that it can relieve the problem of congestion on each LAN. Typical modern LAN interfaces are capable of generating traffic at 10% or more of the LAN bandwidth. Although most LANs can have 100’s of hosts attached, in a workstation environment with file servers, this traffic can limit the practical number of hosts to 10s. If broadcast, or even multicast, is heavily used by all hosts, for instance for location of services, this will lead to heavy congestion.

By dividing a LAN with gateways or bridges, these problems can both be alleviated, since an intelligent router will only forward traffic with a non-local destination.

6.5 Routing and Loop Resolution in Internetwork routers

Internetwork addresses are commonly structured into *network part* and *host part*. Internetwork routing algorithms use the network part to identify to where a packet should be forwarded. Most internetworks are based on connectionless protocols. An important consequence of this is that it is possible to employ fully distributed dynamic routing algorithms. The best known example of this is in the DARPA Internet where internetwork routers (here referred to as gateways)**[Hin83a]** employ the Shortest Path First algorithm to calculate their routing tables, and periodically (every minute or so) exchange these tables with their nearest neighbours. A modification of this is for the gateways to exchange their tables with all other known gateways, although this entails far more network traffic.

In parallel with this gateway-gateway exchange, a reachability or "hello" protocol traces the availability of neighbour gateways and links, and this is used to update local tables to calculate the minimum spanning

tree.[Dij59a] Current research work is trying to solve problems with scaling and stability of such protocols and algorithms. Fluctuations in connectivity can result in temporary loops in the topology, since different routers will discover these fluctuations at different times and construct spanning trees that disagree.

One method of limiting the traffic in large internets is based on hierarchical routing. If extra information is present in addresses, for instance geographic location, it is possible to limit the scope of a search for, and use of alternate routes when a link or router goes down.[Mil84a] Another area of work is in the extension of the metrics used to choose paths. Current routes are chosen on the basis of hops. In LANs, this will commonly be adequate, although where the LAN technologies differ greatly, it may be necessary to avoid loading some intermediate LANs with through traffic. In contrast, the inter-LAN router may have static routing tables. In this case, it is simply an administrative task to isolate loops, and set up routes. Usually, as in the case of the Universe ring-ring bridges, this is achieved by booting the bridges with static routing tables.

6.5.1 Loop resolution in MAC level bridges

Existing LAN-LAN bridges often restrict the topology to exclude loops. Star and/or chain topologies may be allowed. An organisation with a large collection of LANs may wish to benefit from possible redundant routes, for reasons of reliability and load sharing.

Some LAN-LAN bridges use very simple learning schemes to determine which hosts are on which LANs. There is a choice here.

- Forward all packets immediately, and rely on the difference in time of arrival of duplicates to determine loops and best routes, and to build up lists of Hosts on directly attached and remote LANs.
- Accumulate local host addresses before forwarding, and then negotiate with other bridges which hosts are on which LANs.

The first of these schemes has many problems. It may take some time to allow forwarding of packets, and may make fault finding very hard. It could cause problems for higher level protocols by generating large numbers of duplicates, and excess traffic. The second of these schemes requires a bridge-bridge protocol to be designed, and this is what we address below. Here we present an algorithm which LAN-LAN bridges can use to unambiguously learn which hosts are on which LANs, and decide routes between LANs.

The aim of this algorithm is to present a protocol that can determine automatically which hosts are on a particular network (LAN). This is basically an extension of the *learning* algorithms used for current MAC level bridges. When a bridges starts up, it listens on its network interfaces to learn which hosts are on the network. All hosts discovered in this way are recorded and marked as possibly being on the network where they are first seen. At this point no forwarding is done. Next the bridge searches each of its directly connected networks for other bridges. Bridges found in this way will be neighbour bridges. Once another bridge is located then the two bridges exchange the lists which they have learned, and the location of other bridges which they have discovered. At this stage the bridges agree to allocate each network a unique network number, probably a random number seeded from the Ethernet address of one of the bridges. If other bridges confirm that a particular host is on a network then that host is marked as definitely on that network. Also if no contradiction is discovered then an entry is marked as definite. If no other bridges are discovered on a particular network, then all of the entries which were learned as definite are marked, and a number allocated for that network. Once the hosts are marked as being definitely on a network, start forwarding packets for these hosts. All forwarding is of course restricted by the normal MAC bridge access control schemes.

Each entry in the list is assigned a *time to live* field which is decremented by a regular time event. Once this has reached zero then the entry in the list is considered to have timed-out, and is marked as probable again. At this point the negotiation that happened earlier is repeated. The advantage of timing out the entries in this way is that the algorithm is resilient to hosts moving from one network to another. The negotiation period should be small enough that any higher level protocols should not time-out. This would imply that this procedure should not take longer than say 10-15 seconds, however in practice it should take nowhere

near as much time as this.

Once the bridges have determined the network topology, and have knowledge about host location, traditional routing algorithms used in internetwork routers can be employed to forward the packets.

6.6 Remote access to Management Information

Since internetwork routers operate in a protocol specific environment, it is common to choose protocols to access management information from that environment. For instance, IP routers often allow operators to make terminal connections, and interrogate or alter routing tables and access control lists, or to list traffic and error statistics. Agreement about protocols for program access to such information is further away. One reason for this is that terminal access can be authorised by passwords, whereas mechanisms for authentication of programs are not widespread yet.

In contrast to this, MAC bridges contain no protocol implementations, and there are no obvious choices for remote operator or program access to management information. Often, simple serial line access to a bridge is provided, which does not provide any simple means of integrating management access into the rest of the network architecture. Another problem here is the fact that MAC bridges operate in promiscuous mode, and are in some sense anonymous or transparent. To allow a MAC bridge to be addressed individually involves adding some bridge identification. Since solutions to loop resolution problems also involve adding such identification (see above), and entail a bridge-bridge protocol, the design of host-bridge management access protocols should be combined with the design of these bridge-bridge protocols.

6.7 Performance Considerations

In interconnecting LANs via a 100 Mbps network, we must consider expected traffic patterns to ascertain the bridge performance constraints.

A single Ethernet is capable of generating 14000 packets per second of minimum size packets. This is an unlikely statistic since hosts generally compete for the Ethernet, and more usually contention results in maximum utilisations of around 30-40%. A single FDDI network might be capable of 100,000 packets per second.

Measurements of bridged LAN systems at other sites have found typical inter-LAN traffic to be of the order of 30% of intra-LAN traffic. Thus we might expect some 1500 packets per second per Ethernet, as typical average (busy time of day). The current nodes can almost handle this, however that is with a single LAN interface per node. If this was increased to the full population of 4 per board, we would be operating at 25% of requirement implied above (assuming the current bottleneck is CPU in the forwarding node). However, peak loads may be far higher than the average. Typical fileserver accesses are very bursty. For instance, Sun NFS uses bursts of 8 fullsize Ethernet frame packets at full Ethernet packet rate. A lab with 20 teaching machines accessing a single server over the backbone would therefore peak at 160 frames back to back.

We should not just consider the throughput of the bridges in bits, or packets per second. Many protocols are sensitive to variance in delay over the network, and some are even sensitive to slight increases in delay (NFS and DECNET/LAT exhibit this behaviour).

Since backbone and MAC Bridging technologies are essentially transparent to hosts as far as routing and addressing is concerned, it is hard for the network managers to configure their hosts to adjust protocol parameters to operate well regardless of whether destinations are local or remote over the backbone, since this would have to be configured on a per destination MAC address basis. We may expect there to be several thousand hosts on the backbone within 5 years, so this is not acceptable.

The consequence for performance is that delay should be minimal, and that delay should vary smoothly with load. A more subtle constraint is that inter-frame arrival times should be approximately maintained, and bursts of frames should be kept together as far as possible, although one would expect a scattering of the inter-frame times as load increases. Clearly, this has an impact on queueing strategies in the nodes.

6.8 ISDN and Bridges/Connectionless Routers

If we are to interconnect LANs and MANs using primary rate ISDN or Broadband ISDN, we must devise a scheme for controlling the timing, duration and number of calls between sites, depending on the types of traffic currently between two or more sites. We envisage constructing a special kind of router. Based on our experience of routing datagrams over X.25, we believe that we can build a device that matches single packets to ISDN calls.

Essentially, when a packet arrives, a lookup is done on its NSAP (or L-SAP depending on level), which matches to an ISDN destination. If a call does not yet exist, one is set up. (If the delay is too long, we rely on higher levels to recover - this is not counter to the philosophy of datagram protocols, which generally have this assumption, although we may need to feedback some information to the end user). If a call exists, the packet is forwarded to that call. If the number of packets arriving for a given call in a sample period exceeds some threshold, another call is made to that destination. If this is not possible, the appropriate (random) packets are dropped, and the sources informed. If no packets arrive for a destination within some sample period, a call is removed.

Careful measurement of a variety of applications shows that it is feasible to identify classes of applications by the pattern of packets they generate at the network and link level. It is then possible to build a simple recogniser in the LAN/MAN/ISDN router for these patterns. They are then mapped to the appropriate thresholds and sample periods using for call creation and deletion. The cost benefits are clear above the use of leased line services, since we are sharing the cost with all other ISDN users while not using the network. Measurements show that traffic between remote sites is generally in bursts of longish duration, with long gaps between, it is feasible to take advantage of the ISDN charging strategies in several ways. For instance, we can tune bulk transfer applications (such as file transfer and E-mail) to sort their output queues by application on the basis of destination. This then means that batches of calls may be made. It may also be feasible to third party route traffic between MANs over multiple ISDN hops where there are existing calls open with bandwidth to spare, thus saving per call charges. This would depend on the regulatory controls!

6.8.1 Telecomms and Datacomms

Many research programs now exist in the area of Integrated Communications Services. We can broadly classify services as traditional telecom and datacom. By this we mean:

- Real Time

  These include voice and video. They are characterised by the *timeliness* with which data must be delivered, and by the fact that data loss is tolerable within well defined constraints.

- Flow Controllable

  These are characterised by the fact that end-to-end data loss is not tolerable, but sources may be flow controlled or back pressured within well defined limits.

There is a grey area between these services which includes two very important classes of application:

1. Compressed Video which has some of the characteristics of data communications, in that the *rate* at which data is offered to the network may be controlled by altering the compression ratio (at the expense of quality).

2. Remote Windowing protocols which have many of the characteristics of real time services, in that high variance of delay is not tolerable to the end user.

The purpose of current research in Broadband ISDN (especially under RACE) is to identify how the network may support both these kinds of services, and still maintain high utilisation. The most promising start is the use of ATM (Asynchronous Transfer Mode) in the network, instead of the more traditional synchronous protocols. We still require routing, congestion and flow control Algorithms that can mix and match these services. It is also important to identify how much is to be done within the routing nodes in the network, and how much in the (customer premises) terminal equipment.

6.9 Related Work

Bridging is described in the 802 report if the IEEE. Early bridge designs include Waters' and Leslie's for the Universe project.[Wat82a]

Crowcroft[Cro85b] describes the implementation of a high speed router on a multiprocessor architecture (68000 multibus backplane).[Cro85b]

As LAN Interconnect becomes the main role for the long haul networks, the mismatch between communication latencies and throughput for local and wide area communication are rapidly decreasing. In this chapter, we have outlined the current interconnection strategies (employed on the Admiral project, and on the JANET II UK National Academic network). These are important for understanding the end system protocol requirements and recovery techniques, which can be much more lightweight than in the past, where delay, throughput and errors were orders of magnitude greater in the long haul than in the LAN.

In the next chapter, we will look at a low level network access control scheme which a very important component of distributed system spanning more than a single administration.

7. Chapter 7: LAN Access Control at the MAC level

This chapter is about the kind of security mechanisms which are feasible at the MAC level on Local Area Networks. As stated in the introduction, security is a vital part of any distributed systems architecture. The contribution presented in this chapter is by no means a complete solution. However, it represents one step in a bottom up approach to improving the security of systems such as bridged LANs in common use in medium sized institutions (e.g. Campus Networks). We accept that high levels of security include mechanisms such as link level encryption, end to end encryption, originator/recipient authentication and possibly even random background traffic generation. Our embedded approach to constraining LAN access paths is a small, lightweight part of a larger jigsaw.

We outline the types of attack that concern LAN users at this level, and examine the ease or difficulty of implementing these attacks on two common technologies. We then outline a collection of mechanisms to prevent these various attacks, and attempt to estimate their relative costs. Finally we examine some of the special problems created by LAN Interconnection.[30]

Appendix 5 describes the Admiral network on which much of this system was implemented.**Cro88c**

7.1 Introduction

By analogy with the X.400 security model, the kinds of security we might be interested in for a LAN are:

- Trust that the apparent sender of a message is the actual sender.
- Trust that only the intended receiver receives the message.
- Proof that the message is the one sent, and not altered somewhere on the wire.
- Proof that the message is not a replay of an earlier one.
- Confidence that the message is not captured by anyone else on the wire.

In addition to these requirements, we might have stronger requirements that:

- The pattern of messages between hosts is not detectable.
- No host can deny communication between other hosts (perhaps a weaker version of this might be that any pair of hosts will be able to communicate within some deterministic time).

The two most common LAN architectures are Ethernet (Broadcast Bus) and Token Ring.

In the next subsection we look at physical attacks on these two systems.

7.2 Physical Attacks

Ethernets are particularly prone to physical attack due both to the passive nature of the system and the method of attachment. It is feasible to put a new tap onto an Ether Co-ax while the system is running, and almost all effects while this is being done are indistinguishable from *normal* events on an Ethernet. A very expensive TDR (Time Domain Reflectometer) allows detection of new taps, but not with a high degree of accuracy, and usually only when the network is disabled.

It is much harder to attach hosts to a Token Ring, since each station is active, and the ring must be broken to add a new station. It is still feasible to attach a new host to an existing station (if the host is sufficiently sophisticated) and not be noticed. It is trivial for some special monitor station to be designed to detect new stations.

_____

30. "What we have here is a failure to communicate", the Sheriff to Paul Newman in Cool Hand Luke.

In both cases, the data rates are sufficiently low to mean that even a modest host can be programmed to intercept packets, although in the case of token ring, it requires modification to hardware.

7.3 Lan Interconnection

Because of the open nature of MAC bridges, their main advantage for protocol transparency can also be their main disadvantage for network security from the point of view of access control. The model that the FDDI/Ethernet Bridges present is that the FDDI network is a *backbone* which interconnects a number of Ethernets transparently. Frames are forwarded via the Bridges and the backbone between one Ethernet and another. LAN frames are encapsulated in FDDI frames rather than being simply forwarded. We are not concerned with whether the frames on the backbone propagate to all Bridges or are forwarded only to the correct bridge, except in how this might affect performance of the backbone.

The addition of access control filters based on well known information in LAN frames being forwarded is seen as the only way in which traffic may be restricted between certain end points, or from using intermediate networks (LANs) as transit carriers.

The next section describes the simplest useful scheme for LAN frame filtering. The following section describes a more general scheme. The final section describes what remote management should be provided for these mechanisms.

7.4 Access Control Filters

This section lists the basic set of access control filters that we would like to be implemented in the Ethernet/FDDI Bridges. The simplest piece of information that is fairly difficult to forge, and readily available is the LAN address of a host. The source address of a host on a LAN is already the basis for traffic localisation, via the learning mechanism in the DIUs (FDDI Ethernet interface boards).

1. Exclusion.

   Specific hosts should be excluded from access to the backbone, by listing their LAN address in a table of barred hosts. This is so that misbehaving hosts, or hosts that have no good reason to place traffic on the backbone do not place undue load on the network.

2. Inclusion.

   All hosts should be excluded from access to the backbone, with the exception of specific privileged hosts, listed by LAN address in a table. This is so that one can more easily manage LANs with large numbers of hosts, but few with privileges/reasons to use the backbone.

3. Cooperation.

   Pairs of specific hosts should be allowed to exchange frames via the backbone by listing their LAN addresses in another table. This is for the case where pairs of highly secure hosts wish to communicate via the backbone, but do not want traffic from any other hosts. It is also useful when there are non-secure hosts on LANs which have the capability of altering the LAN addresses easily (e.g. DECNET, and some proprietary PC communications systems).

4. Dissociation.

   Pairs of specific hosts should be excluded from exchanging frames across the backbone, by listing their LAN addresses in a fourth table. This is so that hosts which may disrupt the backbone, or LANs connected to it, can be prevented from doing so. A simple example of this is hosts that have aberrant/disruptive or non-standard protocols that conflict with other protocols on other LANs.

In addition to these facilities, it should be possible to bar certain specific *frame types* (in the D/I/X Blue Book sense) from being forwarded to or from a specific LAN. This is useful as a blanket mechanism to prevent certain protocols from crossing to LANs where they may interfere. For instance, Appletalk or DECNET may use similar addresses, or may broadcast frames that should be restricted to a single LAN to avoid conflicts.

7.5 General Filters

There are already two major standard frames for Ethernet LANs: Blue Book and IEEE 802.3. The difference in these is that BB treats the 13th and 14th bytes of the frame as a frame type field, while IEEE treats it as a length. These can safely be distinguished, since all legal lengths are less than the Ethernet maximum frame size (1512 octets), while all legal types are greater than this. However, this means that filtering of protocols based on this field is not always possible (e.g. in the case of OSI or Pink Book protocols, the IEEE standard is adhered to).

We would propose a more general filtering mechanism to deal with this problem. It should be feasible to filter frames by *Arbitrary octet sequence* absolutely anywhere within a frame. The rules for filtering should be at least as wide as those described in the previous section, i.e. allow forwarding of frames with specified sequence, or disallow frames with specified sequence. We might go further, and say that a set of masks and matches should be configurable to check for certain bit sequences within frames to decide whether to forward or bar them. This extreme measure may be costly (in terms of CPU and memory in the Bridges), but is the only way to make this mechanism proof against future protocol changes.

7.6 Remote Management

It should be possible to manage remotely the lists and masks and matches held in each bridge , via some standard protocol, across the LANs and backbone from any secure host, via some standard protocol, across the LANs and backbone from any secure host. Remote access to such management functions should be protected by at least a password mechanism, and should possibly involve encryption of the management data, since Ethernets are very easy to tap.

Such remote management should also be able to retrieve existing tables from the Bridges. It should also be feasible to base the management on more user friendly mechanisms for specifying hosts, such as host names, although this is likely to have to be tied to the higher level protocol suites being run in the hosts. The system should also be able to group hosts in some logical fashion, and specify access control on the basis of entire LANs (e.g. in the case where there are LANs bridged together, before being bridged to the backbone, it should be possible to exclude second or third hop LANs from backbone access if required). The mechanisms described should be sufficient to allow the coexistence of secure and non-secure LANs attached to the backbone, and even simultaneously using it.

The mechanisms should *not* impose any overhead (e.g. increase forwarding delay or loss of throughput) when not in use, and only minimal overhead when used reasonably. [Implementation experience at UCL shows that most of these functions are feasible, by simply enhancing the existing learning process in Bridges rather than complex code in the forwarding process].

7.7 Denial of Service

Although traffic analysis is not relevant to the environments we are considering, denial of service may well be. For example, a malicious user may cost a company or University time, effort and money if they disrupt normal communications for any significant period. Aside from the simple physical threat of breaking the medium, we must consider misbehaviour in the protocols.

In the case of CSMA/CD, it is perfectly possible for two apparently correct implementations to prevent any access to the network for any other hosts (analogous to livelock). If these hosts alternate transmissions rapidly enough, they will hold all other access to a CSMA/CD net in their listen before send step - a loose analogy could be formed with phase locking. They would not be breaking the protocol rules, and therefore would not set off any alarms (a normal host could not tell the difference between two and many hosts causing a busy network). This is a serious flaw in its design as regards security against this form of attack.

In this case of token ring, each station has guaranteed deterministic access (eventually), thus only non-conformant stations can cause a problem. If they are non-conformant, it should be feasible to detect them with suitable monitoring equipment. Obviously, over active stations may deny some bandwidth to other hosts, but this is degradation of service, not denial.

7.8 Access Control

There are two main mechanisms for MAC level bridging:

1. The "Spanning Tree" algorithm (IEEE 802.1 Addendum D).

   The spanning tree mechanism works by learning. Bridges exchange messages with each other to discover a single spanning tree of the graph of interconnected LANs. They then learn which hosts are downstream of which LAN interface ("port").

   The hosts protocols need no change. This is favoured for interconnection of CSMA/CD Broadcast networks.

2. Source Routing

   Hosts wishing to communicate with hosts on other LANs have to form a source route, which is a list of all the LAN addresses of intermediate bridges, and put this in every frame they transmit.

   Host protocols must change. This is favoured for Token Ring interconnection.

The former mechanism is not subject to attack through a station masquerading as a bridge. The latter might be when a genuine bridge was not present.

7.9 Management

It is likely that network managers will maintain access control lists in a central database, for convenience. This database will be downloaded into bridges (at boot time). This download protocol is particularly vulnerable, and must be protected by authentication and encryption. It may be feasible to use secret key cryptography, since this is naturally a centralised system in any case.

One special area for concern is the management of multicast and broadcast protocol access between LANs. A problem can be illustrated by describing an incident with the Appletalk protocols: when two separate LANs running Appletalk were bridged, clients of print servers found their output appearing on the print servers on the opposite LAN.

The problem here is that Appletalk uses a negotiation ("claim/challenge") mechanism to allocate addresses at many protocol levels. If two LANs hosts are booted separately, they are extremely likely to allocate identical addresses for all hosts. When the LANs are connected (or reconnected after failure/partition), there is no mechanism for renegotiation based on detected clashes - indeed there is no mechanism for detecting clashes, despite the fact that the original negotiation, as well as the server location protocol are broadcast based, and therefore open. This means that we need to restrict broadcast (multicast) on a per protocol basis, so that well behaved users of broadcast are not restricted. Another problem with broadcast services is simply that of traffic growth and synchronisation. This has been addressed elsewhere.**Wat86a**

7.10 Routing.

Spanning Tree routing as described above, is incapable of forming more that one single spanning tree over the set of bridged LANs.**Per84a** This means that traffic from all stations on LAN A to LAN B always follows the same route (as does the reverse path traffic unless very peculiar weights are used in deciding "best path"). Source routing is capable of using a spanning tree for each host. However, this entails large amounts of configuration for each host. It might be possible to use a special protocol (akin to the "proxy ARP approach described in [RFC925, Postel]) to perform route discovery, but there is no standard for this yet.

At a completely different protocol layer Network level relays have been used more successfully to restrict routes according to some policy. It is also a more natural place to place this kind of access control. MAC Bridging is adequate for small to medium scale systems (indeed, the standard limits the network diameter to 7 hops); however, overloading MAC bridge functionality with complex topology management is probably a bad design.

7.10.1 Bridge Trust

Even though we do not advocate bridges performing policy routing, it may be important for bridges to trust each other. A station that masqueraded as a bridge could deprive hosts of service. It might even be possible for a station which pretended to be the "root" bridge of a Spanning tree system to completely partition all the LANs. Bridges tend to communicate using multicast. As with the downline load protocol, it may be sufficient to authenticate bridges to each other, and protect their PDUs using a private key cryptographic system. For hosts to trust bridges (and vice versa) it is necessary to employ the techniques described in section 3.

7.11 Policy Routing Spanning Tree

Currently, FDDI nodes operate with the fibre as a switching fabric. The backbone nodes encapsulate non-local LAN frames in the private FDDI frames, and forward them to all other nodes. These then filter on the basis of their local learned lists (forwarding database).

The preferred approach is to MAC bridge between LAN and backbone, which would involve treating the backbone as a two hop MAC route, and running spanning tree (IEEE802.1 Addendum D, now revision E). Each bridge participates in the exchange of configuration BPDUs to form a single spanning tree of the LANs and backbone (i.e. the LANs and backbone are all peers). The way this functions is roughly as follows.

- Each bridge takes a unique ID and port number for each LAN interface [typically based on the hardware address].
- It sets its own address to be root, and lowest port number to be root, and distance from root to zero.
- It sets the state of each port to LISTENING.
- It broadcasts this, once per "hello time" on each of its LAN interfaces (actually multicasts to the "well-known bridge multicast address) with the root port and id it knows about, and the distance to root.
- When it receives hello messages, it decides whether these come from a better suited root bridge. If so, it sets the root bridge to that one, and distance to those seen in the message, and continues sending this as its current information on all other ports, and sets the port this better hello message arrived on to BLOCKING.
- After a number of hello times, it sets non-blocking ports to LEARNING (and starts to accumulate local lists).
- After a number more of hello times (to avoid topology changes during learning affecting things) it sets its learning ports to FORWARDING, and starts the job.
- Whenever it *fails* to get a hello on any port, it essentially restarts this algorithm on that and any other blocking ports...
- Each bridge (backbone node) learns about MAC addresses upstream and downstream. This entails running the bridge FDDI interfaces in promiscuous mode, or else all FDDI nodes broadcasting (multicasting) to all the other FDDI nodes.

  Arriving frame source addresses are stored in the port database, while arriving frames with destination addresses in another port database are forwarded to that port (if it is in forwarding state). This entails treating LAN addresses and FDDI addresses as exactly equivalent, or else providing a globally unique mapping between any different size addresses.

The main advantage of this scheme is that frames are *only forwarded to a single bridge on each hop*. Another advantage is that the attachment of hosts direct to the backbone merely requires a standard FDDI device driver for whatever protocol suite they run, and they are then visible to hosts on all the attached LANs and other hosts on the FDDI.

The main disadvantage is that all forwarding goes towards the root, and then down, and so we expect for a single LAN attached to the root FDDI node, all other traffic *between* LANs has to transit the fiber twice, from source LAN/FDDI node, to root, then from root to destination LAN/FDDI bridge.

The tradeoff is in favour of this scheme for some sites, if there is a logical place to configure as root bridge, say a Computer Centre node, since much traffic will either be destined there, or else transit there to remote sites. An standard extension to this algorithm is to specify a weight for each LAN port. The weight is added into the distance calculation when choosing between two otherwise equivalent paths:

Now, an extension to the idea is to include a notion of policy in the routes.

If a number of hosts wish to exercise some policy over the set of LANs they will allow or disallow traffic to transit, then two things must happen:

1. The hosts must indicate this by 'colouring' their packets appropriately. An inelegant (but practical) way of achieving this (for experimental purposes) might be to use different frame types for different policies. Ideally, we would extend the frame format to include a packet colour field. This need only be 8 bits, since the maximum extent of bridged LANs is usually diameter 7 (i.e. O(100) LANs in all), and we would not expect a large number of different policies.
2. The bridges must recognise this colour, and match it to an appropriate route.

Currently, the Spanning Tree algorithm only produces one global tree - i.e. all packets go by the same routes. A simple extension is to multi-home the ports of each bridge:

For each policy expressed by a host that mentions a LAN, each bridge on that LAN must invent a new pseudo-port for that LAN. If the policy is a inclusion one, then in some sense, the weights must be adjusted downwards to favour this LAN in routes formed by the spanning tree. If exclusive, then we set the state of this port to DISABLED. How can this be done in a distributed manner?

1. For exclusion, it is only necessary for the host group excluding a particular LAN from its routes to indicate to all immediately attached bridges on that LAN that there is such a policy. They then create a policy port, and DISABLE it.
2. For inclusion of a LAN, there are two approaches. If only one LAN is involved, then we simply make a bridge on that LAN the root of the spanning tree for this policy. If more than one LAN are involved, then we start at the host group (assuming the group is on a single LAN - it is a trivial extension if the group is split over several LANs). We query all bridges on this LAN as to the distance (sum of weights) to the root. We then tell the bridges attached to the included LAN to set their weights (on this policy port) to this LAN low enough (or negative) to make this LAN favourable in the spanning tree.

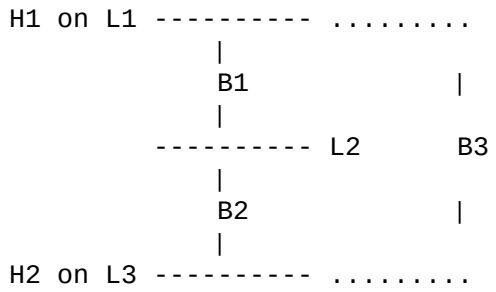


Figure 53. Example 1 LAN Topology

Consider the topology shown in Figure 53: Let us assume that the bridges IDs, ports and weights are such that the normal default spanning tree forces h1-h2 traffic across L2.

If H1 specifies that its traffic must NOT transit L2, then it sends its packets coloured with this policy (say #1). B1, B2 and B3 invent an L2', and thus have two extra ports which are disabled. At the same time, a positive policy must be formed so that B3 will actually now forward packets (normally it would disable its ports). So we invent an L1' and an L3', for which B3 will now route so long as B2 and B3 are disabled for it. In fact, it appears that L3' is not attached to B2, rather to B3 only.

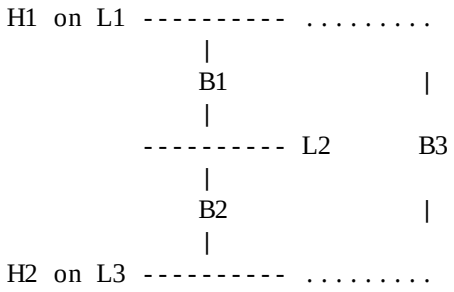


Figure 54. Example 2 LAN Topology

Now, looking at the topology in Figure 54: Let us assume that the bridges IDs, ports and weights are such that L1 to L3 traffic normally routes via B3. Now if H1 and H2 wish to force their traffic (say for load balancing reasons) through L2, we need to look at the cost through B3, and invent an L2' in B1 and B2 which has lower cost (perhaps negative). It then appears to B1 that H2/L3 is nearer on its L2' port, and vice versa for the return path.

More generally, most routing schemes form spanning trees: the 802.1D is no exception. The extension we propose suggests a more general problem. Given a set of LANs and Bridges, what weights do we allocate LANs (ports) such that a given LAN (port) is (is not) included in the set of routes between a group of hosts on 1 or more LANs (i.e. one or more logical LANs). It is a matter for further work to determine a general algorithm for allocating weights (and minimising the number of extra logical LANs introduced) in a distributed way.

The problem with the simple version of the scheme presented above is shown by a third example, displayed in Figure 55:

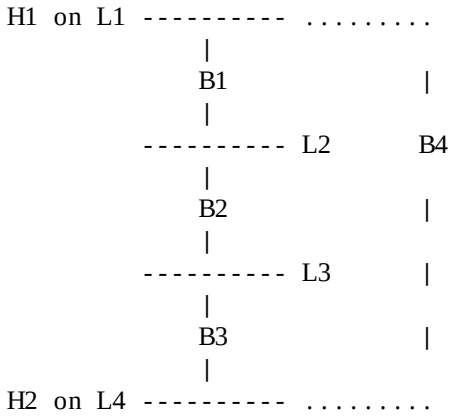


Figure 55. Example 3 LAN Topology

To disallow traffic from H1 to H2 to go over either L2 or L3 is the same as before. To force traffic over L2, or L3 or both involves inventing new logical LANs/ports in all the bridges, and adjusting all their the weights. We have yet to devise an algorithm for this.

7.12 Costs of Policies.

The basic management cost is that each bridge attached to an included or excluded LAN must have a copy of all the port state for each policy.

The spanning tree steady state cost for exchange of Bridge Configuration packets is known to be constant - i.e. one per designated bridge per hello time. If we have to run a set of policy spanning trees, we can look at this as a set of logical LANs. The overhead is then linear in the number of policies - i.e. one hello per hello time per designated bridge per policy. The forwarding cost is one extra lookup per packet, based on the packet colour, top find the correct spanning tree (actually the correct LEARNED host list for this policy spanning tree).

Obviously, to form a number of different spanning trees, there should be a number of different physical topologies that achieve the correct reachability. This is a matter for the network designer to get right. The costs of our scheme are relatively modest, and we believe that it achieves a reasonable degree of security for these costs. The most important assumption is that the bridges are all trusted.

7.13 Related Work

Further study of policy routing can be found in Eliasz and Crowcroft.**Eli9.a, Eli90a**

Clearly, extending bridging to include more complex access control is only one limited approach to adding important security features to a distributed system. In the final analysis, End System security is the only true protection against illegal access to resources. However, we believe that the mechanisms outlined in this chapter can provide a small part of an overall system for restricting access to network resources, and providing some degree of protection against end system attacks by increasing the cost/complexity of mounting such an attack (e.g. forcing the perpetrator to spoof an existing MAC address, which can often be easily detected).

8. Chapter 8: Conclusions and The Future

In this chapter, we summarise what we believe to be the contribution we have made in this thesis, and outline what extensions would be important.

8.1 Conclusions

In chapters 1 and 2, we described our model of the environment that most computer communication will operate in over the next decade as shown in Figure 56:

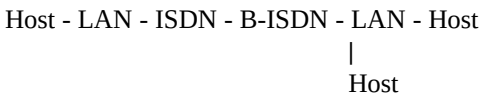


Figure 56. Computer-Computer Communications Model

The building blocks for future communication are shown in Figure 57:

wire ->checker->sequencer->flow controller->double checker->aggregator->process

wire <-formatter<-pacer<-collector<memory/process

Figure 57. Communications Functional Building Block Model

The network is a statistical Multiplexing shared resource. End systems must implement End-to-End protocols to provide the service needed by an application. However, in terms of implementation, the network *should* only provide a best effort delivery, otherwise it cannot extend the maximum range of quality of service while maintaining near optimum sharing. In contrast, hosts *must* provide full error recovery to applications that need it.

In terms of the concurrency in end system protocol implementations, the thesis in this work is that it should be limited to 3 levels: The application itself, the end-to-end protocol, and the network level. Any other levels should be implemented inline to avoid context switching overheads, and unnecessary critical region protection.

Chapters 3, 4 and 5 form the core of the thesis, and are original work on end-to-end, or transport protocols to support distributed computing.

8.1.1 Request/Response Protocols

Chapter 3 presented the design and implementation of a Sequential Exchange Protocol to support distributed processing paradigms such as RPC and reliable messaging. The original contribution that this protocol makes is that its design (and implementation) are simple, and yet flexible enough to support RPCs and reliable streamed messages efficiently over both LANs and WANs.

8.1.2 Multicast

In chapter 4 we have presented the design of a new transport protocol which supports an n-Reliable multicast Request Response type service. Initial implementation experience shows that this kind of protocol can reduce the number of packets on the network, whilst more conveniently providing a similar type of service to sequential unicast requests and responses. Analysis supports this experience for LAN use of the protocol. Based on this analysis, the protocol implements a multiple coupled windowing scheme of flow control for n to one communication, which is designed to solve the problem of overruns in both a client host, and in intermediate nodes in an internet.

The implementation and use of voting functions requires further investigation. The request response layer should not have to buffer all the parts of all the replies until the vote function has been applied. This is why we introduced a message level vote function. This is in some way derived (hopefully automatically) from the higher level vote function, so that the client message layer may cancel unwanted replies as soon as possible in an exchange. This derivation may be non-trivial, and so for the present, the vote function may only be a simple comparison of some portion (or all) of some number of reply messages, which returns the list of addresses of servers whose replies are acceptable.

8.1.3 Transport Protocol Performance

Being able to adapt to loss, delay variation with load, congestion and so forth is an essential part of a good transport protocol implementation in a connectionless internet. Some initial TCP implementations had fixed timeouts and simplistic adaptive windows. Usually these were optimal for a low delay, low error rate, uncongested LAN. In chapter 5, we described measurements over SATNET, either alone, or further on into ARPANET, that show that the combination of changes made to the BSD43 TCP have vastly improved TCP performance, from previous throughputs of 3-4 Kbps, up now greater than 12 Kbps. The real gain is in reducing the number of unnecessary retransmissions, which has dropped effectively to zero. The main gain in throughput for a single connection is the increase in window size, and for multiple connections is due to the fair sharing caused by the congestion window mechanism. With the existing TCP protocol (i.e. sliding window, go back n retransmission) we would get somewhat better throughput if the error characteristics of SATNET were better (e.g. less than 1% packet loss through error) since the congestion window would not operate unnecessarily so often. Without changing the protocol (and to some extent the entire Internet architecture), it is impossible to tell the difference between loss through congestion, and loss through genuine channel error.

It was observed that the characteristics of SATNET in terms of delay-bandwidth product match those of the high speed experimental networks (e.g. FDDI) that were emerging as the new technology in the late 1980s. SATNET was an invaluable environment for investigating protocol behaviour over such paths, since the actual data rates are slow enough that the quantity of data that must be collected to characterise such behaviour is much lower than that required for a high speed, lower delay network. It is also easier to timestamp events accurately without the requirement for special clock hardware.

8.1.4 Future Transport Protocol Enhancements/Measurements

Future work includes the addition of a possible selective acknowledgement/retransmission option to TCP. **Jac90c, Jac88b** and some mechanism to help TCP to deal with SATNET's random loss characteristics.

Another area worth exploring is rate controlling the IP. One way for information to get to IP from the network without requiring too much protocol complexity (i.e. acknowledged IP) is to use the existing mechanism of ICMP source quench messages, which are sent by gateways when their queues start to get overloaded.**Pru87a** One advantage of this mechanism is that it is independent from the transport protocol, and could therefore potentially apply equally well to UDP (or RDP or VMTP or ESP etc etc). However, there are some arguments against this: Under fairly weak assumptions on the underlying network, it is possible to prove that window flow control is optimal. I.e., of all possible controls it is the scheme that gives best throughput. [31]

---

31. The End-to-End task force/research group argue: The way optimality is proved also suggests that trying to "force congestion out to the fringes and keep the interior uncongested" (rfc1016, p.13) is wrong. What you want to have is a backlog of traffic at the bottleneck so that it can keep its link utilization high independent of source propagation delay and "noise" effects on bottleneck-inbound traffic. The farther the bottleneck is from the source, the larger its queue needs to be to absorb incoming traffic dead times. I.e., if "congestion" is "queue length", it wants to be in the center for performance, not at the edges. This scheme will scale badly since, under random traffic patterns, reservation schemes must leave resources committed but idle for a fair fraction of an e-e prop delay. Thus the flow control "overhead" in such networks (bandwidth lost to the flow control scheme) increases linearly with diameter. Or, to put it another way, bandwidth available to users decreases linearly with diameter.

    Stability is assured by that fact that Window Flow Control is conservative: it cannot send a new packet into the network until an old one exits (at equilibrium). Thus arrival rate is explicitly clamped to departure rate (= service time = bandwidth of slowest link in path/bottleneck).

    Rate based Flow Control saves network bandwidth by replacing the acknowledgement "clock" with a local timer. The information

"Experience implementing protocols suggests avoiding timers in protocols whenever possible." If there are many hosts connected to the same network running SQuID. under moderate to heavy load they should all equilibrate to approximately the same D value, i.e., all hosts on the network will start to generate packets at about the same frequency. Once the frequency is tied down, the only degree of freedom left to the network is to vary phase.

If the traffic frequency is random, it is easy to see that small changes in the frequency will make small changes in the throughput. I.e., X(f) is continuous, where X is average throughput and f is the frequency you inject packets. If traffic and your frequency is fixed, X will depend on phase. Unfortunately, X(p) is not only discontinuous, the discontinuity is at the optimum. The best and worst phase are infinitely close together and a small change in p can make a large change (the largest possible change) in X(p).

Practically, this means that feedback systems that involve fixed frequencies (where phase has meaning) are very difficult to control. The normal hill-climbing and gradient procedures (e.g., adjust by some fraction of an error signal) no longer work: they tend to drive the system into a limit cycle, oscillating between the best and worse states. Adaptive rate-based-flow-control is a fixed frequency system and I don't think stable, effective controls for it are going to be easy to come up with (they exist, I am sure, but phase controls tend to be subtle and hard to find).

Some measurement needs to be done to gain further understanding of any second order effects due to different delay distributions for large data packets in one direction and small acknowledgement packets in the reverse (e.g. if a different number of connections were running in each direction, what would happen?).

8.1.5 Hardware

In appendix 8, we described some of the possible areas for hardware support of the protocols described in chapters 3, 4 and 5. The End-to-End argument is applied yet again to show how only pipelined support for features of transport protocols may be implemented in a front end processor. The extension of packet switching from the network into the actual Bus of the end system seems a promising generalisation.

8.1.6 Routing and Interconnect

More and more sites are finding single LANs insufficient for their local communication requirements. Conventional repeaters simply allow more complex topologies of LANs, without extending their geographic scope. In chapter 6 and 7, we compared and contrasted connecting LANs together by two very different means. Where LAN technology is sufficiently similar, we conclude that there are no special reasons not to interconnect LANs by MAC level bridges, provided sufficiently clever loop resolution. Where protocol independence is not required, then internetwork level routers are sufficient, and existing routing schemes will eliminate loops.

8.1.7 FDDI and Broadband ISDN. Interconnection

We have seen the interconnection of LANs over slow speed WANs. More recently, we have looked at higher speed LANs and Wide Area facilities. We have been involved in a number of projects concerned with interconnection of networks. Currently, we are interested in the interconnection of LANs, MANs and various varieties of ISDN. Within the college in the late 1980s, we had some 25 Ethernet LANs, a number of proprietary LANs, and a backbone FDDI Network, with four nodes (and one standby) connecting a number of the Ethernets together. Within the Department, we have access to a number of Wide Area Networks:

- The UK Academic X.25 network, JANET. This is used for all standard data network services in the Academic community.

---

content (clock info) of the acks must still be derived and that will require additional state be kept at the endpoints. You can save bits on the wire by putting state bits at the endpoints but total information is conserved.) The IP layer is not the right place for a lot of state.

- The Public X.25 Data networks.
- A number of leased line (including a switched 2 Mbps service) for research projects. The Admiral project (Alvey funded) investigated call control and charging aspects of switching over a wide area between LANs.
- Direct connectivity to the US Internet (via NSFnet). This allows the research communities in the UK and US to exchange mail and files. UCL's direct connection has allowed protocol performance experiments.
- A Pilot ISDN service. This was provided both for service and to test the feasibility of building Ethernet/ISDN routers.
- A Fiber network that connects most of the London University Colleges both with analogue video, and digitally at 2 Mbps. This is mainly used for Video Conferencing and Seminars.

We have been concerned in researching ways of connecting these various networks together, to provide some integration of services. We have not done a great deal of work in the area of voice switching yet. The use of FDDI as backbone technology is functionally very useful, however we should like to transition to a full MAC bridged system (possibly with multiple spanning trees to load share) as standards become more established.[32] Access control in a bridged network system in our operating environment is essential.

Performance of the FDDI should in principle support up to 30 typical LANs, given experience of traffic patterns, but engineering for overload at peaks must be very careful so as not to upset existing protocol implementations. Measurements of the FDDI/Ethernet nodes at the end-to-end host level is encouraging in the short term, but future performance requirements will be far more severe.

8.1.8 Theoretical Considerations

Appendix 9 presented some theoretical considerations of designing and implementing protocols for distributed systems. The contribution here is the initial steps in the demonstration of equivalence between a specification of minimally layered (network and transport) systems to a required reliable service. The bounded buffer abstraction is sufficient for most applications in terms of end-to-end reliability and flow control. Other layers are best implemented as pipelines of simple data transformation (including checksumming, encryption/decryption and presentation coding/decoding).

8.2 Future Work

Over the last 2 years, digital audio and video devices have become available at cost comparable to workstations. This means that the workstation can function easily as a video-phone on the local network. It will be necessary for the wide area network to be significantly re-engineered to support protocols that can handle the timeliness requirements of these newer services. It remains to be seen if this current architecture can be extended or must be re-built from the ground up. In this section I describe some of the work currently in progress and indicate how the research in this thesis is being carried forward.

8.2.1 Future Continuous Media Work

In 1990, the Department of Computer Science, University College London, commissioned what we believe to be the world's first Multimedia TeleConferencing system over International Packet Switched Networks. This work marks the start of the next phase of development of protocols for Distributed Systems: Protocols for Distributed Multimedia Systems.

The UCL Conferencing Facility is described in a review article **Cro1.a** Research following this thesis is continuing in the following three areas:

---

32. A set of acceptance tests carried out by the author on the Bloomsbury Computing Consortium FDDI network, in 1988, are described in appendix 9.

1. The Communications Architecture & Protocols:

   Workstations commonly use the TCP/IP protocols. The **ST protocol** [Top90a] is an extension of these to support packet video and voice. Some research is going on in how to make conventional Internet systems also support real time traffic. However, future networks will be based on **Asynchronous Transfer Mode** (ATM) -[Gha85a] this has impact on workstation network interface and protocol support designs.[Smi91a]

   Video Coder/Decoders (**CODECs**) not only provide Analog to Digital conversion, but also highly efficient image compression, both for compact storage, and timely transmission. They will soon be incorporated in workstation hardware.

2. The impact of real time protocols on the Workstation Architecture and the Operating System.

   If video and voice is carried in ST we must support this in the workstation. We also need support for video & audio device I/O. Alternatively, if we are running our teleconferencing over an ATM network, we must support ATM network interfaces. If we need image compression due to lack of bandwidth, this needs to be integrated with image capture, display and transmission.

3. Integration of video and voice with text and graphics on the workstation screen involves adding functionality to *Windowing Systems*.

8.2.2 Communications Architecture

Protocols in End Systems (host computers/workstations) and Intermediate Systems (Routers/Bridges) both need to be modified to support video and voice traffic as well as ordinary data:

1. They must support low variation of bandwidth, delay and error for video and voice traffic[33]
2. Multicasting packets[Dee85a]
   (and multicast forwarding in routers) increases efficiency of use when supporting more than 2 way conferencing over LANs and WANs, by up to *n* times for *n* participants. Since video requires high bandwidth, this is important.
3. Traffic from different streams may belong to the same conference and thus needs synchronising (not just *lip-synch* but mouse pointer movement and other data must be synchronised with video/voice).

8.2.3 ST versus IP or CLNP with multicast

The communications architecture UCL that is currently using is based around the DARPA Internet Protocol suite including not only the extensions described in this thesis, but also with certain further enhancements to support *real time* traffic.

These are provided by the ST protocol, running in Butterfly gateways for end and intermediate systems, or in Sun Sparcstations for end systems. To provide video compression so that we can run our conference over typical wide area networks, analogue video is converted into compressed digital video by a CODEC. The ST gateway then runs two further protocols to carry the video and voice:

1. Packet Video Protocol/PVP
2. Voice Traffic protocol/NVP

---

33. Delay, in principle, must vary by less than 1/2 a frame time for video (e.g. 1/60 or 1/50 a second) and 1-200 msecs for voice. Bandwidth can vary if it results in loss, but only up to a few 10s of percent.

    MPEG proposes both forward and backward prediction with reference points at least 5 frames apart - this gives 6 frames of delay in both the encoding and decoding processes.

Each of these is tuned to the special requirements of video and voice.

ST is a variant of the Internet Datagram Protocol (IP). The main difference is that it provides resource reservation facilities for PVP and NVP to declare requirements in advance. A control message is sent using the Stream Control Message Protocol to setup paths through the network from the source of a conference site to all the destinations ("Multicast"). This message carries a "FlowSpec" to all the ST/IP[Par91a] gateways which says what the delay/bandwidth/error requirements of the requester are.

In fact, the ST protocol specification has little to say about how the "FlowSpec" is enforced or how multi-destination delivery is achieved - it is largely left to *underlying mechanisms*. Because of this, we are investigating whether the mechanisms for loose resource reservation and multicast can be implemented for conventional IP hosts and routers, so that we can run on almost un-modified parts of the Internet.[34] Clearly a Connection Oriented Network Service would be an alternative choice, so long as the CONS suppressed retransmissions which would cause undue (and unnecessary) delay variations over long haul lines. It would also be necessary to have multicast facilities and synchronisation points in such a CONS stream.

8.2.4 Asynchronous Transfer Mode

Some underlying technologies provide multicast and resource reservation already. One of the emerging standards for future wide area networks is the Asynchronous Transfer Mode of the Broadband ISDN service. This is highly suited to video and voice traffic (not surprisingly, as it is developed by Telecom Companies as a future replacement for the worldwide phone system). This is achieved mainly through using small, fixed size packets. If we can attach our workstation to such a network, it will have a lot less work to do to maintain the characteristics needed for video and voice. On the other hand, it may have to do a lot more to run ordinary data traffic as we shall see below.

8.2.5 CODECs, Varying the Bit Rate

Recently there has been a lot of work on standard image compression techniques. We now have JPEG for static pictures and MPEG (from ISO) and H.261 (from CCITT) for moving picture storage and transmission.[Gal91a] Chipsets for these are being developed rapidly, so that the cost of an Analog to Digital (A2D) converter which can do image compression may be very low within a year or two.

Moving image compression has two main components of interest:

1. Intra-frame compression, where each frame/image is separately compressed (using something like DCT).
2. Inter-frame compression, where some kind of image differencing is done (this is highly efficient for video without scene changes).

There are interesting interactions between packet loss and image de-compression algorithms when inter-frame compression is in use. Some inter-frame compression techniques still send an independent "re-synchronising" frame (intra-frame compressed) to anticipate problems like this.[35]

8.2.6 Multi-media Workstation Architecture

Once the network is in place to carry traffic around with real time characteristics, we need to get this traffic in and out of the workstation. As well as keyboard and mouse, we need camera and microphone (and possibly scanners). There are then various choices as to paths for analogue and digital data through the workstation, and onto the network, together with appropriate places to perform A2D conversions, and possible picture (and audio) compression/decompression:

---

34. IP could easily be replaced with the ISO CLNP, if there was a standard for multicast. This is now a working item in the relevant committees, based on IP multicast model.

35. The present UCL system uses a proprietary CODEC. A CCITT version will shortly become available. We believe it will be feasible to place different CODECs on a single analogue switch and relay video this way.

1. Video display on a window on the workstation screen. (Cards are available to do this now on many popular workstations).
2. There are two further levels of integration. One is to put the CODEC on a board in the workstation, and have it talk to the analogue video controller, with some other channel to interact with the windowing system.
3. The second is to have the digital video stream be converted straight into the frame buffer (in the correct place), and avoid the use of a CODEC for conversion between the network and on-screen. The CODEC is still needed for camera to the network. Indeed, we would need the CODEC to talk through a high(-ish) speed serial link to the workstation, which would then packetize and synchronize this with audio and other signals.

8.2.7 TCP/IP and ST support

ST/IP from a set of remote sites and reverse path traffic are processed as follows:

- ST from remote sites arrives at the Ethernet interface (or high speed serial if that connects to remote site), and is de-multiplexed from IP up to ST to the PVP and NVP protocol modules in the workstation.
- The ST stream then needs to be decoded by a packetizer and the CODEC. This can be done by running the packetizer code as a kernel process, feeding the output(s) to the high speed serial link out to the CODEC which then runs the analogue signal to the video card input (which then talks (out of band) to the windowing system, and puts the video in the frame buffer in the appropriate area, and thus on the screen.
- The reverse path involves analogue signal from the camera (or a switch) to the CODEC.
- Then the compressed coded traffic goes through the high speed serial input to the workstation, and is packetized.
- Then its put through PVP, and ST/IP and out onto the network.
- Audio is similar, but makes use of the audio hardware on the workstation to do the coding/decoding, and then maps this to a standard audio encoding through the NVP process, and thence to/from the net.

The current video protocol stack is illustrated in Figure 58.

Output	Input
Analogue Video Camera	Video Card
CODEC	CODEC
Serial Line	Serial Line
Packetizer	De-Packetizer
PVP	PVP
ST	ST
IP	IP
Ether or Serial Line	Ether or Serial Line

Figure 58. Packet Video Protocol Stack

The audio protocol stack is illustrated in Figure 59.

Output	Input
Analogue audio mike	Speaker
Signal Proc chip	Signal Processing Chip
NVP	NVP
ST	ST
IP	IP
Ether or Serial Line	Ether or Serial Line

Figure 59. Packet Audio Protocol Stack

When using ordinary packet switched protocols to carry the video/voice traffic, we need to be able o schedule the processing of arriving packets so that they can be delivered to output devices smoothly (we cannot **overrun or starve** a loudspeaker or framebuffer). This Real Time support in operating system, is rather an anathema to many versions of Unix: Either all the work must be in the kernel or we must change the scheduler.

8.2.8 Workstation ATM support

When the network is delivering many small fixed size cells, video can be digitised (uncompressed) directly from a camera into the network, with the workstation simply providing call setup and teardown. Inbound video could be received direct into the workstation framebuffer. In this case, we no longer need the full real time support for the video and voice streams. However, we do need means to synchronise events between the video stream and data (mouse movements).

8.2.9 Compression - MPEG on Workstation

We envisage MPEG chips or the like being available in the same way as audio, very soon (sooner than ATM networks). The minimum effect of having a low cost compressed digital video I/O device will be to make wide area networked multi-media more accessible, and local area more popular. Until we have ATM, however, the workstations and routers will have more work to do to maintain the delay/bandwidth/error characteristics. Video hardware will not obviate the need for clever routers and scheduling.

8.2.10 Shared Windows and Shared Views

Many people have started to add the facilities to networked windowing systems **Get86a** to share views of normal data (text/graphics) on screen sharing X.Han92a Some work has been done on adding in video windows. **Bru89a, Hom90a** Some window managers support extended views of a workspace.

To share the output window from an X Client to many workstations is reasonable easily achieved, and with a reliable multicast transport protocol, can be done efficiently.[36] Essentially, the X Client makes contact with an X redistribution server which in turn contacts all the participants servers. Some out of band management (part of floor control) mechanism chooses who has input (which server keyboard/mouse events reach the client) at any one time (or there may be a free-for-all).

Choosing where to add digital video to a networked window system involves several choices:

1. Is the video sent by an ordinary client as a sequence of window bitmaps of some kind to the display server, using the ordinary protocol?

36. An application could be shared by taking the command language or interface to the application, and distributing that rather than the window's view on the application. If the application does file or database access, one could even share the access at that level, and replicate the application itself. The disadvantage of this approach is that one then needs to implement some type of transaction processing to avoid consistency problems on access to shared data. The advantage is increased concurrency. Sharing a terminal emulator is so straightforward and allows all programs with a text based command language to be trivially shared, that the more application specific approach seems unnecessary.

2. Is there an extension to the windowing protocol to carry video (possibly encoded using MPEG)?
3. Is the video client/server a separate (but integrally controlled) system, due to its continuous/real time nature?

We are interested in how we can combine these facilities with the scheduling necessary in Unix and with window management, and more importantly, with **Floor Control** and **Navigation**. Floor control is essentially concerned with who has input control to the shared workspace. Navigation is mainly concerned with finding who & what are available to share workspaces. We are devising a combination of these facilities to form what might be termed a Cyber Space windowing system [Gibson].

8.2.11 Distributed Multi-media Systems

Re-designing networks, workstations, window systems and operating systems to support real time multi-media conferencing is not easy. None of these components was designed to meet the requirements in the first place. We believe that re-engineering in situ is hard, but still possible. We also believe that the trend away from heavily layered systems towards application layer framing and cut through implementation of most protocol data unit handling is supported at least in part by the work presented in this thesis.

9. References

Some references are to documents called RFCs are *Request for Comment* and are kept on-line at a number of Network Information Centers, and information servers. RFCs are published by the SRI NIC. Others are to documents published by official standards bodies such as ISO under UN, or CCITT under ITU, or national bodies. These are normally available through the national standards body, such as BSI in the UK.

References.

Ada83a. C J Adams, GC Adams, AG Waters, I Leslie, and PT Kirstein, ‘‘Protocol architecture of the UNIVERSE project’’, *Universe Project Report*, pp. 379-383 (1983).

And91a. Andrews, G.R., *Concurrent Programming. Principles and Practice*, Benjamin Cummings, ISBN 0 8053 0086 4 (1991).

And82a. GR Andrews, ‘‘The Distributed Programming Language SR - Mechanisms, Design and Implementation’’, *Software - Practice and Experience* **12**pp. 719-753 (1982).

Bac86a. B Bacarisse, ‘‘Admiral Remote Procedure Call: User Guide for Release 2 and 3’’, Internal note IN-1936, Department of Computer Science, University College London (April 1986).

Bac8.a. B Bacarisse, J Crowcroft, M Riddoch, and S Wilbur, " ‘‘The Design and Implementation of a Protocol for Remote Procedure Call", ’’, *Proceedings UK IT ’88,* , (Swansea, July 1988.).

Ben83a. E Benhamou and J Estrin, ‘‘Multilevel InternetworkingGateways: Architecture and Applications’’, *IEEE Computer, Vol.16, No. 9, pp. 27-34*, (September 1983).

Ben82a. CJ Bennett, ‘‘The Overheads of Transnetwork Fragmentation’’, *Computer Networks, Vol. 6, No. 1, pp. 21-36*, (February 1982).

Ber85a. JA Berntsen, ‘‘MAC Layer Interconnection of IEEE 802 Local Area Networks’’, *Computer Networks and ISDN Systems, 10, pp. 259-273*, (1985).

Bir85a. K Birman, ‘‘Replication and Fault Tolerance in the ISIS System’’, *ACM Operating Systems Review* **19**(5) pp. 79-86 (Demeber 1985).

Bir87a. Kenneth P. Birman and Thomas A. Joseph, ‘‘Reliable Communication in the Presence of Failures’’, *ACM Trans.Comp.Syst.* **5**(1) pp. 47-76 (Feb 1987).

Bir84a. Andrew D. Birrell and Bruce J. Nelson, ‘‘Implementing Remote Procedure Calls’’, *ACM Trans.Comp.Sys.* **2**(1) pp. 39-59 (Feb 1984).

Bog80a. DR Boggs, ‘‘PUP: An Internetwork Architecture’’, *IEEE Trans on Comms, Vol. COM-28, No. 4, pp. 612-623*, (April 1980).

Bra85a. R Braden, ‘‘Transaction Protocols’’, *Request for Comment* **955**, SRI International (September 1985).

Bru89a. Todd Brunhoff, ‘‘Pleasing to the Eye’’, *Unix Review*, (Oct 1989).

Cas90a. Case, J.D., Fedor, M., Schoffstall, M.L., and Davin, C. , ‘‘Simple Network Management Protocol (SNMP). ’’, 1157 (May 1990).

Cel83a. N Celandroni, ‘‘STELLA – Satellite Interconnection of Local Area Networks: Architecture and Protocols’’, *Proc. IFIP Int. Symp. Satellite and Computer Communications, North-Holland, Versailles, pp. 25-40*, (April 1983).

Cha83a. J. Chang and N.F. Maxemchuk, ‘‘Reliable Broadcast Protocols’’, *ACM Trans. Comp. Systems.* **2, 3**pp. 251-273 (Aug. 1983).

Che86a. D. Cheriton, ‘‘VMTP: a transport protocol for the next generation of communication systems’’, *Computer Communications Review* **16**pp. 406-15 (5-7 August 1986).

Che85a. D R Cheriton and Willi Zwaenepool, ‘‘Distributed Processes in the V-kernel’’, *ACM Transactions on Computer Systems* **3**pp. 77-107 (May 1985).

Che86b. D. R. Cheriton, ‘‘Request-Response and Multicast Interprocess Communication in the V Kernel’’, *?*, (post Aug 1986).

Che90a. Greg Chesson, *XTP Specification rev. 3.5*, Protocol Engines inc. (September 1990).

Cla82a. D Clark, ‘‘Modularity and Efficiency in Protocol Implementation’’, RFC817 (July 1982).

Cla82b. DD Clark, ‘‘Window and Acknowledgement Strategy in TCP’’, *RFC 813, M.I.T. Laboratory for Computer Science*, (July 1982).

Cla85a. DD Clark, ‘‘NETBLT: A Bulk Data Transfer Protocol’’, *RFC 969, M.I.T. Laboratory for Computer Science*, (December 1985).

Cla90a. David Clark and David Tennenhouse, ‘‘Architectural Considerations for a new Generation of Protocols’’, *ACM SIGCOMM90*, , ACM (September 1990).

Coh81a. D. Cohen, ‘‘On Holy Wars and a plea for peace’’, *IEEE Computer magazine*, (October 1981).

Col86a. Robert Cole and Peter Lloyd, ‘‘OSI Transport Protocol’’, pp. 33-43 in *Proc. Conf. Open Systems 86*, Online, London (March 1986).

Col83a. R. Cole, C. Adams, N.Calandroni, L. Lenzini, and I. Leslie, ‘‘An interworking architecture for STELLA and UNIVERSE’’, *Satellite and Computer Communication*, pp. 369-377 (1983).

Com88a. D. Comer, *Interworking with TCP/IP, Principles, Protocols and Architecture*, Prentice Hall International, ISBN 0 13 468505 9 (1988).

Cor81a. Xerox Corp., ‘‘Courier: The Remote Procedure Call Protocol’’, *XSIS 038112*, , Xerox Corp. (Dec 1981).

COS84a. COST, ‘‘Network Interconnection: Principles, Architecture and Protocol Implications’’, *Final Report of COST 11 BIS Local Area Network Project, Part 1*, (November 1984).

Croa. J Crowcroft and K Paliwoda , " ‘‘Reliable Multicast Protocols", ’’, *Proceedings of the Alvey 88 IT Conference.*, .

Cro85b. J Crowcroft, ‘‘Admiral HSWAN/Ethernet Bridge User Guide’’, *Admiral Note*, , Admiral (1985).

Cro85a. J Crowcroft and M Riddoch, ‘‘Sequenced Exchange Protocol’’, *UCL Internal Note 1824, ADMIRAL Project Note A.341*, (1985).

Cro86a. J Crowcroft, ‘‘Protocol implementations for CMOS’’, *UCL Internal Note 1998*, (1986).

Cro88b. J Crowcroft and M Riddoch, " *Managing the Interconnection of LANs" Proceedings of IFIP Conference Berlin*" .if !""" ," . 1988.

Cro88c. J Crowcroft and M Riddoch, " ‘‘High Speed Interconnection of LANs"’’, *Proc Alvey 88 IT Conference*, (July 1988).

Cro88a. J Crowcroft and K Paliwoda , " ‘‘A Multicast Transport Protocol",’’, *Proceedings of the ACM SIGCOMM 88 Symposium*, (1988).

Cro1.a. J Crowcroft, P Kirstein, and D Timm, " ‘‘Multimedia TeleConferencing over International Packet Switched Networks", ’’, *Computer Communications*, , pp. Butterworth (April, 1991.).

Cyp78a. RJ Cypser, ‘‘Communications Architecture for Distributed Systems’’, *Addison-Wesley*, (1978).

Dal70a. Y. Dalal, ‘‘Broadcast Protocols’’, *Stanford University Ph.D Thesis*, (1970).

Dan80a. A Danthine and F Magnee, ‘‘Transport Layer - Long-Haul vs. Local Networks’’, *Proc. Int. Workshop on Local Area Networks, IFIP WG 6.4, Zurich*, (August 1980).

DEC82a. DEC, ‘‘DECNET Digital Network Architecture NSP Functional Specification’’, *Phase IV, Version 4.0.0., Digital Equipment Corporation,* (March 1982).

Dee85a. S. E. Deering and Dave E. Cheriton, ‘‘Host Groups: A Multicast Extension to the Internet Protocol’’, *RFC-966*, pp. 1-27 (Dec 1985).

Dij59a. Dijkstra, ‘‘A Note on Two Problems in Connexion with Graphs’’, *Numerische Mathematik* **1**pp. 269-271 (1959).

ECM82a. ECMA, ‘‘Local Area Networks - Layers 1 to 4: Architecture and Protocols’’, *ECMA TR/14*, (September 1982).

Edg83a. SW Edge, ‘‘Connection Flow Control in a Wide Area Packet Switching Network’’, *Ph.D. Thesis, TR 88, University College London*, (September 1983).

Edg84a. SW Edge, ‘‘An Adaptive Timeout Algorithm for Retransmission across a Packet Switching Network’’, *Proc. Communications Architectures and Protocols Symp., SIGCOMM ’84, ACM, Montreal, pp. 248-255*, (June 1984).

Eli9.a. A Eliasz, S Bavan, and J Crowcroft, " ‘‘Adaptive Network Management Using Neural Computing", ’’, *3rd RACE TMN Workshop,* , (Heathrow, August 1989.).

Eli90a. A Eliasz and J Crowcroft, " ‘‘Some Investigations into Policy Based Routing Schemes and Systems"’’, *International Symposium on Local Communications Systems Management,*, pp. North Holland IFIP, (Sep, 1990).

Fel90a. Feldmeier, D and McAuley, D, ‘‘Ordering Constraints in Communications Protocols’’, *Proceedings ACM Sigcomm 90 Symposium.*, pp. ACM (September 1990).

Fra85a. A. J. Frank, L. D. Whittie, and A. J. Bernstein, ‘‘Multicast Communication on Network Computers’’, *IEEE Software*, pp. 49-61 (May 1985).

Fri85a. M Fridrich and W Older, ‘‘Helix: The Architecture of the XMS Distributed File System’’, *IEEE Software* **S-2**(3) pp. 21-29 (May 1985).

Gal91a. Didier Le Gall, ‘‘MPEG: A Video Compression Standard for Multimedia Applications’’, *Communications of the ACM* **34**pp. 47-58, ACM (April 1991).

Get86a. Scheifler, Robert and Jim Gettys, , " ‘‘The X Window System," ’’, *ACM Transactions on Graphics,* **vol. 5, no. 2, pp. 79-109** (April 1986).

Gha85a. M Ghanbari, ‘‘Two-Layer Coding of Video Signals for VBR Networks’’, *IEEE Journal on Selected Areas in Communications* **7,5**pp. 771-781, IEEE (June 1985).

Gif85a. D Gifford and R Baldwin, ‘‘An Architecture for Large Scale Information Systems’’, *ACM Operating Systems Review* **19**(5) pp. 161-170 (December 1985).

Hal88a. Fred Halsall, *Data Communications, Computer Networks and OSI*, Addison-Wesley (1988).

Han92a. Mark J. Handley and Prof Steve R. Wilbur, *Multimedia Conferencing: from prototype to national pilot.* 1992.

Hin83a. R. Hinden, J. Haverty, and A. Sheltzer, ‘‘The DARPA Internet: Interconnecting Heterogeneous Computer Networks with Gateways’’, *IEEE Computer* **16**(9) pp. 38-48 (September 1983).

Hoa78a. CAR Hoare, ‘‘Communicating Sequential Processes’’, *Comm. ACM* **21**(8) pp. 666-677 (August 1978).

Hom90a. Anderson, DO, Govindan, R, Homset, G, ‘‘Abstractions for Continuous Media in a Network Window System’’, *Berkeley Tech Report 90/596* **596**, UCB (1990).

Hug86a. L. Hughes, ‘‘A Multicast Transmission Taxonomy’’, *Technical Report Series no. 221*, pp. 1-15, Newcastle University (Aug 1986).

Hug88a. L. Hughes, ‘‘A Multicast interface for UNIX 4.3’’, *Software Practice & Exp.* **18**(1) pp. 15-27 (Jan 1988).

Hut88a. Hutchinson, N. and Peterson, L., ‘‘Design of the x-kernel’’, *Proceedings of SIGCOMM 88*, pp. 65-75 (August 1988).

IEE83a. IEEE, ‘‘Special Issue on Open Systems Interconnection (OSI)’’, *Proc. IEEE* **71**(12) pp. 1329-1488 (December 1983).

IEE85a. IEEE, ‘‘LAN Interconnection using MAC Sublayer Bridges’’, *IEEE 802.1 Internetworking Task Group, 802.85*1.188*, (July 1985).

Inc90a. Protocol Engines Inc, *The eXpress Transfer Protocol Specification, rev 3.5*. September 1990.

Inc86a. Sun Microsystems Inc, ‘‘Remote Procedure Call Protocol Specification’’, *rfc1057*, (February 1986).

ISO83a. ISO, ‘‘Connection Oriented Transport Protocol Specification’’, *ISO 8073*, (November 1983).

ISO84a. ISO, ‘‘Basic Reference Model’’, *ISO 7498*, (1984).

Jac88b. Jacobson, V. and Braden, R.T. , ‘‘TCP extensions for long-delay paths. ’’, 1072 (1988).

Jac88a. Van Jacobson, ‘‘Congestion Avoidance and Control’’, *Proceedings ACM SIGCOMM Symposium*, (August 1988).

Jac90b. Van Jacobson and et al, ‘‘TCPDUMP(1), BPF...’’, *Unix Manual Page*, (1990).

Jac90c. Van Jacobson, Bob Braden, and Lixia Zhang, ‘‘TCP Extension for High Speed Paths’’, *RFC 1185*, , DARPA (October 1990).

Jac90a. V Jacobson, ‘‘Compressing TCP/IP headers for low-speed serial links. ’’, RFC 1144 (Feb 1990).

Jai86a. R Jain, ‘‘Divergence of Timeout Algorithms for Packet Retransmissions’’, *Proc. 5th Ann. Int. Phoenix Conf. on Computers and Communications, Scotsdale, AZ., pp. 174-179*, (March 1986).

JNT82a. JNT, ‘‘Cambridge Ring 82 Protocol Specifications’’, *Joint Network Team of the Computer Board and Research Councils*, (September 1982).

Lam78a. Leslie Lamport, ‘‘Time, Clocks and the Ordering of Events in a Distributed System’’, *Comm. of the ACM* **21**pp. 558-565 (July 1978).

Lam81a. Ed. BW Lampson, *Distributed Systems - Architecture and Implementation, An Advanced Course*, Springer-Verlag (1981). ISBN 3-540-12116-1.

Laz81a. ed Lazowska, HM Levy, GT Almes, MJ Fischer, RJ Fowler, and SC Vestal, ‘‘The Architecture of the Eden System’’, *Proceeedings of the 8th ACM Symposium on Operating System Principles*, pp. 148-159 (December 1981).

Lef83a. Samuel J. Leffler, William N. Joy, and Robert S. Fabry, ‘‘4.2BSD Networking Implementation Notes’’, Berkley Report (Revised July 1983).

Lei85a. B.M. Leiner, R.H. Cole, J.B. Postel, and D. Mills, ‘‘The DARPA Internet Protocol Suite’’, *Proceedings INFOCOM85*, , IEEE (March 1985).

Les81a. I Leslie, ‘‘A High Performance Gateway for the Local Connection of Cambridge Rings’’, *Computer Laboratory, University of Cambridge*, (June 1981).

Lis84a. B Liskov, M Herlihy, and L Gilbert, ‘‘Limitations of Remote Procedure Call and Static Process Structure for Distributed Computing’’, Programming Methodology Group Memo 41, MIT (September 1984).

Liu86a. Y. Liu, T. B. Gendreau, C. T. King, and L. M. Ni, ‘‘A session layer design of a reliable IPC system in the Unix 4.2 environment’’, *Proceedings of the Computer Networking Symposium*, pp.

120-9, IEEE Comput.Soc.Press (Nov 1986).

Llo86a. Lloyd, P.J., *End to End Protocol Performance over Linked Local Area Networks*, Ph.D Thesis - UCL Technical Report 123, London (December 1986).

Llo80a. PJ Lloyd, ‘‘The Ring and Internet Addressing’’, *UCL Internal Note 1003*, (October 1980).

Mic89a. Sun Microsystems, ‘‘NFS: Network File System Protocol specification.’’, RFC 1094 (March 1989).

Mil88a. Cohrs, Miller, Call, ‘‘Distributed Upcalls: A Mechanism for Layering Asynchronous Abstractions’’, *Proc Distributed Computing Systems*, pp. IEEE Comp Sci Press (June 1988).

Mil85b. T Miller, ‘‘Internet Reliable Transaction Protocol – Functional and Interface Specification’’, *Advanced Computer Communications Corp., RFC 938*, (February 1985).

Mil84a. D Mills, ‘‘Exterior Gateway Protocol Formal Specification’’, *RFC 904*, , SRI International (April 1984).

Mil85a. DL Mills, ‘‘Network Time Protocol’’, ARPA RFC 957 (September 1985).

Moc83a. P.V. Mockapetris, ‘‘Analysis of reliable multicast algorithms for local networks’’, *Proceedings Eighth Data Communications Symposium*, pp. 150-7, IEEE Computer Soc. Press (3-6 Oct 1983).

Moc84a. P.V. Mockapetris, ‘‘A Domain Nameserver Scheme’’, *IFIP WG 6.5 Conference, Nottingham*, (May 1984).

Moga. Mogul, J.C. and Deering, S.E. , ‘‘Path MTU discovery. ’’, RFC 1191 (1990).

Mog87a. Mogul, J.C., R.F.Rashid, and M.J.Accetta, ‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code’’, *Op Sys Review* **21** (November 1987).

Moy91a. J. Moy, ‘‘Multicast Extensions to OSPF’’, Internet Draft, Proteon, Inc. Network Working Group (July 1991).

Nag84a. J Nagle, ‘‘Congestion Control in IP/TCP Internetworks’’, *Computer Communication Review, Vol. 14, No. 4, pp. 11-17*, (October 1984).

Now78a. D.A. Nowitz and M.E. Lesk, ‘‘A Dial-up Network of UNIX systems’’, UNIX Programmer’s Manual, 7th edition, Bell Labs. (August 1978).

Pan85a. F Panzieri and S Shrivastava, *Rajdoot: A Remote Procedure Call Machanism Supporting Orphan Detection and Killing*, Univerity of Newcastle upon Tyne, Computing Laboratory (1985).

Par91a. Escobar, J, Deutsch, D, Partridge, C, ‘‘A Multi-Service Flow Synchronisation Protocol’’, *BBN STC Tech Report*, , BBN (March 1991).

Per84a. R Perlman, ‘‘An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN’’, *Digital Equipment Corporation, IEEE 802.85*1.97*, (1984).

Plu82a. DC Plummer, ‘‘An Ethernet Address Resolution Protocol’’, *RFC 826*, (November 1982).

Pos81a. J Postel, ‘‘Internet Datagram Protocol RFC791’’, *USC/Information Sciences Institute, RFC 791*, (September 1981).

Pos80a. J. Postel, ‘‘Transmission Control Protocol’’, RFC 793, DARPA (September 1980).

Pos80b. J.B. Postel, ‘‘Internet Control Message Protocol’’, RFC 792 (August 1980).

Pou73a. L Pouzin, ‘‘Interconnection of Packet Switched Networks’’, *INWG Note 42*, (October 1973).

Pru87a. Prue, W. and Postel, J.B. , ‘‘Something a host could do with source quench: The Source Quench Introduced Delay (SQuID). ’’, RFC 1016 (July 1987).

Qua85a. JS Quarterman, ‘‘4.2BSD and 4.3BSD as Examples of the UNIX System’’, *ACM Computing Surveys* **17**(4) pp. 379-418 (Demeber 1985).

R.9.a. Wilbur, S. R., Crowcroft, J., and Murayama, Y., *MAC Layer Security Measures in Local Area Networks*. May 1989..

Ram87a. S. Ramakrishnan and B.N. Jain, ‘‘Protocols for Reliable Multicast Over Local Area Networks’’, *Proceedings of the Twentieth Hawaii International Conference on System Sciences 1987* **3**(2) pp. 500, Hawaii Int. Conference Syst. Sci. (6-9 January 1987).

Ran78a. B Randell, ‘‘Reliable Computing Systems’’, in *Operating Systems - An Advanced Course*, ed. R Bayer, Springer-Verlag (1978). ISBN 3-540-09812-7.

Sal84a. JH Saltzer, DP Reed, and DD Clark, ‘‘End-to-End Arguments in System Design’’, *ACM Transactions on Computer Systems* **2**(4) pp. 277-288 (November 1984).

Sch85a. RE Schantz, RH Thomas, and G Bono, ‘‘The Architecture of the Cronus Distributed Operating System’’, *Report*, , BBN (April 1985).

Seoa. K Seo and J Crowcroft, " ‘‘The SATNET Distributed Measurement Experiment", ’’, *Proceedings of the ACM SIGCOMM 88 Symposium*, .

Sho79a. JF Shoch, ‘‘Packet Fragmentation in Inter-Network Protocols’’, *Computer Networks, Vol. 3, No. 1, p. 3*, (February 1979).

Sho78a. J F Shoch, ‘‘Inter-networking naming, addressing & Routing’’, *Proceedings of COMPCON, IEEE Computer Society, pg 72-79*, pp. 72-79, IEEE (fall 1978).

Smi91a. B Traw, J Smith, ‘‘A High Performance Host Interface for ATM Networks’’, *ACM SIGCOMM*, , ACM (1991).

Spe89a. A. Z. Spector, ‘‘Distributed Transaction Processing Facilities’’, pp. 191-214 in *Distributed Systems*, ed. Sape Mullender, ACM Press (1989).

Ste91a. D. Comer, D L Stevens, *Interworking with TCP/IP, Design, Implementation & Internals*, Prentice Hall International, ISBN 0 13 4654378 5 (1991).

Sun75a. CA Sunshine, ‘‘Interprocess Communication Protocols for Computer Networks’’, *TR 105, Digital Systems Laboratory, Stanford University*, (December 1975).

Svo84a. L Svobodova, ‘‘File Servers for Network-Based Distributed Systems’’, *ACM Computing Surveys* **16**(4) pp. 353-398 (Demeber 1984).

Tan81a. AS Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ. (1981).

Ten89a. D L Tennenhouse, ‘‘Layered Multiplexing Considered Harmful’’, *H Rudin & R Williamson editors, Protocols for High Speed Networks*, pp. Elsevier Science Publishers, IFIP (May 1989).

Top90a. Topolicic, C, ‘‘Experimental Internet Stream Protocol Version II’’, *RFC 1190*, , SRI-NIC (Oct 1990).

Tre80a. SW Treadwell, ‘‘Measurement Methods in Packet Switched Networks’’, *Ph.D. Thesis, University College London, TR 65*, (1980).

Vel84a. D Velten, ‘‘Reliable Data Protocol’’, *RFC 908, BBN Communications Corporation*, (July 1984).

Wai88a. Waitzman, D., Partridge, C., and Deering, S.E. , ‘‘Distance Vector Multicast Routing Protocol. ’’, 1075 (Nov 1988).

Waklya. Ian Wakeman, Jon Crowcroft, Zheng Wang, and Dejan Sirovica, ‘‘Layering considered harmful’’, *To appear, IEEE Network January 1992*, pp. 7, psroff -mm (16 July 1991).

Wan91a. Z Wang and J Crowcroft, " ‘‘A New Approach to Congestion Control in Datagram Networks" ’’, *ACM Performance Evaluation Revue/ACM SIGMETRICS.*, pp. ACM (June 1991).

Wat82a. AG Waters and IM Leslie, ‘‘UNIVERSE Bridges: Implementation Specification’’, *UNIVERSE Project Document UP/95.2*, (February 1982).

Wat86a. A.G. Waters, ‘‘The use of broadcast and multicast techniques on computer networks’’, *Conference on Electronic Delivery of Data and Software*, (Publ. No. 69) pp. 45-50, IERE (Institute of Electronic and Radio Engineers) (16-17 September 1986).

Wat89a. Richard Watson, ‘‘The Delta-t protocol; Features and Experience’’, *14th Local Computer Networks Conference*, (Minneapolis, October 1989).

Wec80a. S Wecker, ‘‘DNA: The Digital Network Architecture’’, *IEEE Trans on Comms, Vol. COM-28, pp. 510-526*, (April 1980).

Wil87a. S Wilbur and B Bacarisse, ‘‘Building Distributed Systems with Remote Procedure Call’’, *IEE Software Engineering Journal* **2**(5) pp. 148-159 (September 1987).

Wil87b. S. Wilbur and P. J. M. Polkinghorne, ‘‘Distributed Robust Filestore’’, *Internal Note*, (1987).

Wro90a. Liskov, B, Shrira, L., Wroclawski, J., ‘‘Efficient At-Most-Once Messages Based on Synchronized Clocks’’, *Proc of SIGCOMM 90*, (April 1990).

Zha90a. Zhang, L, ‘‘Virtual Clock: A New Traffic Control Algorithm for PSNs’’, *SIGCOMM ACM Sumposium*, (1990).

Zha86a. L Zhang, ‘‘Why TCP Timers Don’t Work Well’’, *Proc. Comm. Arch. & Protocols Symp., SIGCOMM86, Vermont, pp. 397-405, ACM*, (August 1986).

Zim80a. H. Zimmerman, ‘‘The ISO model of Architecture for Open System Interconnection’’, *IEEE trans. on Communication COM-28, No. 4*, pp. 425-432 (April 1980).

CONTENTS

LIST OF FIGURES

LIST OF TABLES