The Scalability of Multicast Communication

Mark G. W. Jones

Department of Computer Science,
University College London,
London.

Dissertation submitted for degree of
Master of Philosophy.
January 1993.

Preface

I would like to thank my supervisor, Professor Wilbur, for his encouragement and valuable advice during the course of my research. I would also like to thank Soren-Aksel Sorensen and Mike Luck for many useful discussions.

I would like to make a special note of thanks to my parents who supported me for my final year, and to my friends for their support.

Except where otherwise stated in the text, this dissertation is the result of my own work and is not the outcome of any work done in collaboration. Furthermore, this dissertation is not substantially the same as any that I have submitted for a degree, diploma or any other qualification at any other university. No part of this dissertation has already been, or is being concurrently submitted for any such degree, diploma or any other qualification.

Abstract

Multicast is a communication method which operates on groups of applications. Having multiple instances of an application which are addressed collectively using a unique, multicast address, allows elegant solutions to some of the more intractable problems in distributed programming, such as providing fault tolerance. However, as multicast techniques are applied in areas such as distributed operating systems, where the operating system may span a large number of hosts, or on faster network architectures, where the problems of congestion reduce the effectiveness of the technique, then the *scalability* of multicast must be addressed if multicast is to gain a wider application.

The main scalability issue was considered to be packet loss due to buffer overrun, the most common cause of this buffer overrun being the mismatch in packet arrival rate and packet consumption at the multicast originator, the so-called *implosion* problem. This issue affects positively acknowledged and transactional protocols. As these two techniques are the most common protocol designs, it was felt that an investigation into the problems of these types of protocol would be most effective.

A model for implosion was developed which was simulated in order to investigate the parameters of implosion. A measure of this implosion was derived from the data, this *index of implosion* allowing the severity of implosion to be described as well as the location of the implosion in the model. This implosion index was derived by dividing the rate at which buffers were occupied by the rate at which packets were generated by the model. The value may then be used to predict the number of buffers required given the number of packets expected.

A number of techniques were developed which may be used to offset implosion, either by artificially increasing the inter-packet gap, or by distributing replies so that no one host receives enough packets to cause an implosion. Of these alternatives, the latter offers the most promise, although requiring a large effort to maintain the resulting hierarchical structure in the presence of multiple failures.

Contents

Table of Tables

Table of Figures

Glossary

Broadcast

The transmission of a message from an originator to all recipients. Can be considered as a multicast to an ill defined group.

Bus

A parallel communication channel normally used to connect main memory to peripherals

Client-Server

An interaction architecture where a client accesses a remote service. see transaction

Closed Group

A multicast group which does not allow any access by non-group members, except for joining hosts.

CLOWN

Concatentated LOcal and Wide area Network simulator.

Completion Time

Time for the last expected reply to be received in response to a multicast.

Data Integrity

The assurance that the data transmitted is the data received.

Group

A set of related recipients, identified by a shared address.

Hierarchical Forwarding

A method of gathering replies so that the number of replies directed to any host is less than would cause implosion.

Host

A networked computer which supports protocol operation.

Implosion Index

A number which represents the degree of overflow expected from a buffering system, values ranging from 0 to 1.

Internet

A large collection of connected networks running the Internet suite of protocols.

Internet Protocol (IP)

A network level protocol used for routing within the Internet, part of the Internet suite of protocols.

Local Area Network

A network which extends up to 10km.

Metropolitan Area Network

A network which extends up to 50km.

Module

A simulation entity exhibiting specific behaviour.

Multicast

The transmission of a message from an originator to many recipients.

Multicast Address

A sharable address identifying a group.

N-reliable multicast

Refers to a transaction, where n replies are expected.

Negative Acknowledgement

A method of communication, see protocol, a special packet sent to request the retransmission of a missed packet.

Network Buffer

A part of the host which buffers packets between network and bus.

Open Group

A multicast group which allows access to non-group members. see Client-Server

Open Systems Interconnection

A number of international standards for protocols.

Originator

The source of a multicast, may or may not be a group member.

Packet Efficiency

The ratio of data packets transmitted to the number of packets transmitted overall.

Positive Acknowledgement

A method of communication, see protocol, a special packet sent to acknowledge the reception of a message.

Protocol

An agreed method for transferring data.

Recipient

The intended receiver of a multicast, normally a group member.

Reference Model

A model of communicating entities which has 7 levels.

Reliability

The confidence an application has that a packet will be received by the recipients.

Single Site

One of the group members having special responsibility to the group for ordering purposes.

Transaction

A sequence of exchanges where an originator wishes to use a service provided by a server, passing data with the service request, the service then processing that data an returning a reply.

Transmission Control Protocol (TCP)

A byte stream oriented transport level protocol, part of the Internet suite of protocols.

Unicast

Communication with exactly one recipient.

User Datagram Protocol (UDP)

A transport level datagram protocol, part of the Internet suite of protocols.

Wide Area Network

A network which extends over 50km.

Chapter 1
Introduction

Until the 1980's most computer communication could be characterized as being either *unicast* or *broadcast*, unicast being the predominant communication method employed, the one to one nature of unicast corresponding well to the most common usages of communications, such as file transfer and remote terminals. The introduction of broadcast capable networks allowed an application to communicate with *all* other applications which may be reachable by that broadcast. The main feature of broadcast that allows this one to "all" communication is that the broadcast address is recognized by *every* host on a broadcast capable network.

Multicast may be described as a formalization of the broadcast model, communication being with a *group* of applications, each group identified by a shared, multicast address. One feature of multicast supporting this description is the commonality between multicast and broadcast addresses, each being *sharable*, and *location independent* a property inherited from address sharing. Multicast is a formalized model because the group is now defined by its address, whereas a broadcast was to every host regardless of the actual locations of the individual applications. Indeed, broadcast may be considered to be multicast with an ill-defined group, as the actual membership of the broadcast group is not normally associated with broadcast.

Multicast is often employed as a means of achieving application fault tolerance, multiple copies of an application being used to ensure application availability in the presence of failures. Associating this group of applications with a single multicast address allows a simplified interface for communication to be employed, the group membership being abstracted by the multicast address. While fault tolerance is an important facet of multicast, the use of multiple copies of an application also exhibits powerful distribution capabilities for co-operating applications, where the application copies co-operate in problem solving.

1.1 Objectives

The work presented in this dissertation investigates some aspects of the scalability of multicast communication. As communication takes place using some form of protocol or protocols, the main thrust of this work concerns how protocol design affects the scalability of multicast applications. The first objective was therefore to characterize some of the mechanisms that are scalability issues, these mechanisms then being related to a number of protocol designs.

The second objective, once some of the mechanisms affecting scalability were identified, was to quantify scalability, if possible, with a view to develop a measure of scalability. This objective required that the design of multicast protocols be in some way reduced to the basic operating principles, these operating principles thus forming the subject of investigation rather than the potentially large number of variations. A simple model was therefore developed which was used first to identify those parameters that were considered to affect the scalability of multicast, and secondly to chart the effects of these parameters.

A third objective was to develop methods that may be employed to increase the scalability of multicast protocol designs. A number of methods are described which may be employed to mitigate the implosion problem, the methods being independent of any particular protocol design, being intended more as basic templates for scalable communication. Implosion occurs whenever the number and arrival rate of packets exceeds the capability of the recipient to process the arrivals, which may result in packets being discarded if insufficient buffering is available to temporarily store imploding packets, a particular problem for multicast because of the synchronization of replies.

Motivationally, observation of the literature on multicast indicated that scale was treated tentatively by many multicast protocol designs. While it is acknowledged that scale was never intended by these designs to be of prime importance, it indicates that there was a perceived need for the parameters of scale to be investigated. Many of these designs appeared to be unduly limited in the number of group members supported.

1.2 Multicast Scalability

Deering's definition of multicast [De89a] refers to a "*host group* of zero or more members" as forming a multicast group, this group being defined by the shared address associated with it. The most natural interpretation of scalability is therefore the size of this multicast group, and indeed it is with group size that many of the problems discussed later are concerned. The main problem considered is that of *implosion*, where replies to a multicast arrive at the originator at a rate and with sufficient numbers to overwhelm the buffering capabilities of the originator, leading to lost replies. Of course, this view requires that replies are generated and necessary for the protocol to operate. The requirement for replies is discussed later. A number of other factors which may contribute to scalability includes the loss of messages in transit, and the requirements of group management operations and ordering considerations.

Scalability may be viewed as defining the limitations of a protocol design when the various controlling parameters are scaled. Strictly, for a protocol to be scalable, the size of the group should be transparent to the application, that is there should be no performance impact on the protocol dependent on group size, or any of the other parameters discussed later. Also, scalability is not confined to the maximum size of group supported, being more a measure of the range over which the design operates. Compromises in a design which has a large maximum, may be unsuitable for small group sizes. The scalability of multicast, here considered a problem of protocol design, is of concern because of the lack of *limits* implied by Deering's definition. By implication, a protocol must be at least aware of any limitations on its use in order to ensure that an application is able to operate as intended. It is one of the primary aims of this dissertation at least to suggest that a multicast protocol must be assessed for any such limits.

Because of this lack of intrinsic limits, unless a protocol has been individually assessed for its scalability, any change in one of the parameters affecting scale may make the design unworkable, in the context of group size reducing the maximum group supported to less than required for fault tolerance for example. Conversely, if a protocol design was intended to operate within some real time constraints, [Fe90a] providing a good introduction to such issues, the additional work involved in communicating with a larger number of recipients may delay the protocol resulting in an effective failure. This example also illustrates one of the more important aspects of scalability, that a multicast, under certain conditions, may *never* complete, the overall time required to gather all the replies required exceeding some application defined limit.

As will be discussed later, the environment in which a protocol operates may profoundly affect the operation of that protocol even if all other parameters remain unchanged, this environmental aspect of multicast scalability being of importance given the rapidly advancing pace of communication technology. When it is considered that a protocol design has a relatively long lifetime compared to these technological advances, a design must not only operate within the current environment but also anticipate such changes in order for to produce a return on its design and implementation costs. Indeed, the environmental aspect of multicast has a powerful effect on the scalability as will be shown later.

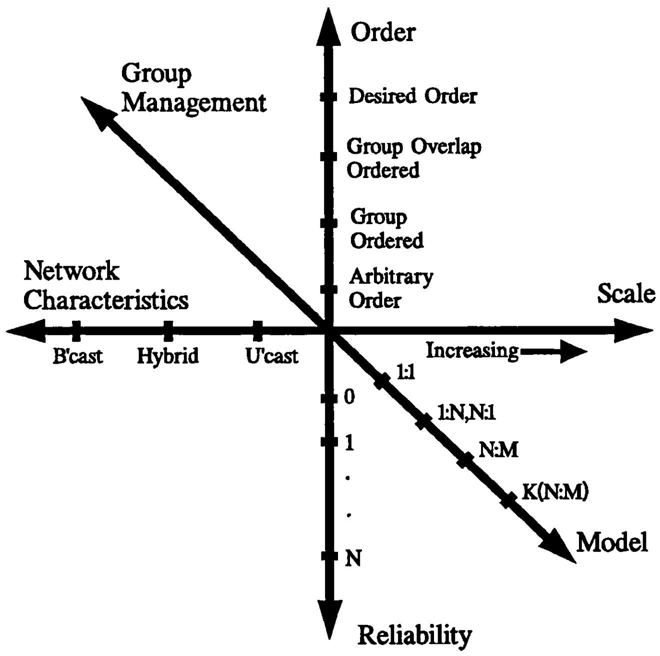


Figure 1.1 Multicast Issues

Figure 1.1 shows how the many inter-related issues of multicast may be illustrated. A few of these issues may be described as environmentally related, such as the underlying network architecture and the structure of the group. The others are considered here to be design choices. The former issues are introduced in this chapter, the latter in the next chapter. As scale is the subject of this dissertation this aspect is discussed later.

The implication of Figure 1.1 is that multicast protocols may not be considered complete without considering the many issues illustrated. The design of ordering systems and group management are influenced by the underlying network architecture and the reliability of the protocol used to transfer information. In turn the reliability mechanism is also influenced by the underlying architecture. The multicast model influences the reliability required as a simple 1:N model has no replies and therefore by implication multicast reliability is not an issue. All of these are influenced by scale in that the use of a particular method to implement an ordering paradigm or group management model may be modified by potential scalability problems.

1.3 Multicast Models

One of these environmental issues is the structure of a group. Four distinct structures are shown in figure 1.1. The 1:1 model represents the unicast case. The N:1 and 1:N models represent the cases where many originators transmit to one recipient, illustrative of a typical multiple client, single file server interaction, and the one to many case which is most often referred to as multicast, respectively. The N:M model is a more general view of a multicast, where many clients are communicating with many members of the same group. Further along this axis, the most general situation is where many sets of such interactions are taking place simultaneously.

These models represent the physical structure of communication. The actual transfer of packets between these models may be bi-directional, so that in each model communication may also take place in the opposite direction to that implied, for example the 1:N model represents a communication of 1:N and N:1, where the N:1 communication is decomposed into N (1:1) communications. This communication model forms the basic model used throughout this dissertation, encapsulating multicast from the point of view of an originator. It is considered that it is the originator that forms the weak link in the scalability of multicast.

In considering communication to be bi-directional, it is assumed that the reply traffic is generated in response to the reception of a multicast, the total forming a complete description of the communication. The types of reply traffic which may be generated can be divided into two broad categories. The first type is used by the protocol for house-keeping functions, such as buffer management and is often referred to as an *acknowledgement.* The second type is used to convey data back to the originator, where the data was generated in response to the forward multicast. This latter type of interaction is called *transactional* and is a common technique in distributed applications.

One further communication model concerns the nature of the group communication, a distinction being made between communication which takes place solely between group members, and communication between group members and non-group members, the former referred to as *closed* group communication, the latter as *open* group communication.

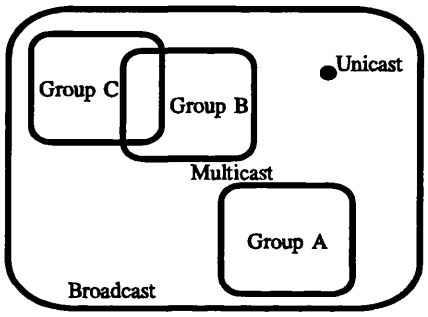


Figure 1.2 Generalized Group Structure of Multicast

Figure 1.2 shows how the various concepts of group fit together, the broadcast group entirely encapsulating the multicast and unicast groups. The notion of overlapped groups, groups B and C, is of significance when order *between* groups is considered. The inclusion of a unicast "point" within the broadcast group leads to the perception that multicast describes a general model for communication.

1.4 Network Architecture

Three types of network architecture are shown in figure 1.1, unicast and broadcast having already been introduced. Network architecture constrains the functionality of a multicast protocol because of the physical nature of the architecture. In the case of broadcast there is no restriction, as by definition any message that is broadcast is potentially receivable by every connected host. Unicast on the other hand forces a multicast to take the form of multiple unicasts. The third type shown is *hybrid,* where the underlying network is less restrictive than unicast, but more restrictive than broadcast. A measure of the efficiency of multicast in these environments is *packet* *efficiency*, which is described as the ratio of multicast packets to the total number of packets transmitted to completion. Mockapetris [Mo83a] formulates the packet efficiencies for a number of common protocol types. In summary, the packet efficiency of a simple protocol which transmits one packet and receives one reply may be described by the formula

$$Packet\ \ Efficiency = \frac{(K+1)}{(K+1)+n} \tag{1.1}$$

where K is the number of additional copies of the packet made, and n is the number of recipients. A broadcast capable network has a value of $K = 0$, as no additional copies are made, a unicast based network requires $K = n - 1$. The n in the equation represents n replies, one from each recipient.

The Internet Protocol (*IP*) [Po81a] forms the network layer of the Internet protocol suite. The Internet architecture is an example of the hybrid style, where packets are routed to their destination by a series of unicast hops. The *IP* possesses a *broadcast* capability as described by Mogul [Mo84a,Mo84b] and Lebowitz and Mankins [Le85a] achieved by copying packets at each routing node rather than just forwarding the received packet, transmitting these copies to other routing nodes. In this way a packet copy is received by each connected host. Because many packet copies are used this is not considered a true broadcast, where one copy is receivable by each host. It is also not pure unicast, as the copying of packets is distributed, and transparent, to the originator. The *IP* style of broadcast forms the basis for the *IP* multicast host extensions described by Deering [De85a,Ch85b,De86a,De88a,De89a].

1.5 Protocol Design Issues

Given the basic structure of the group communication and the underlying restrictions imposed by network architecture, a multicast protocol may be described by its position relative to the remaining axes. Multicast order is a symptom of the grouping of applications, where the actions at each application copy must in some way be related across the group. The four ordering philosophies shown in figure 1.1 are developed in more detail later. Reliability refers to the number of replies received by the originator. The term replies is used crudely here to refer to some form of message which is transmitted in response to the reception of a multicast message. As will be discussed later, such a simple reply is not necessarily observed in practice. Finally, group management is the mechanism employed by a multicast protocol to track changes in group membership. No enumerations are presented as group management may be further sub-divided, which is presented later.

1.6 Multicast Applications and Environments

A multicast application has two properties due to the multicast address. The first is *sharability*, which allows multiple copies of an application to be addressed as a single entity, the second is *location independence* which stems from shareability. By grouping application copies using a single multicast address a number of application properties are more easily achievable. For example, a fault tolerant application may have a number of copies of itself distributed among a number of hosts, all of which receive messages directed to its multicast address. Any application using this service therefore has only to use the shared multicast address to communicate with all instances of the service. Location independence, in addition to allowing the use of multiple application instances, allows a more flexible approach to unicast applications, reinforcing the view that multicast offers a general communication model.

While fault tolerance would commonly employ a few application copies to achieve a particular degree of fault tolerance, the use of large scale multicast in areas such as distributed operating systems, object management and shared memory is becoming evident. While the majority of communication in these environments is unicast or small scale multicast, the possibility that a multicast covers a large number of the nodes in such distributed systems cannot be ignored, for example updating the operating system on each node or monitoring a distributed computation. The use of large scale multicast is also encouraged by the spread of networks, the Internet for example spanning continents and connecting many thousands of hosts. Applying multicast techniques to electronic mail where mail is directed to groups is an example where large scale multicast may offer a reduction in traffic. By extension, computer conferencing could be a major use of multicast in the future, the more real-time aspects of this demanding the type of traffic reduction potentially offered by multicast.

One of the simplest applications of multicast is that of *service location*. By multicasting to the address of a known service, the response of that service may be used to locate, that is get the unicast address of, that service. Using multicast for this is preferable to employing broadcast, as a multicast is directed only to the hosts which support the required service, reducing the number of messages directed to hosts which do not support the required service. As only one instance of a service is required, any extra replies are commonly discarded. The service location model is also applicable to many simple applications, such as a time or name service, where the reply contains the information provided by the service, in the case of a time service, the current time. An example of an application is the Ethernet Address Resolution Protocol [Pl82a] which is used to map an *IP* address to the fixed Ethernet address.

More generally, the multicast models described above are extensively used in distributed operating systems [Ta86a,Mu88a], examples of the use of such models being described more fully later. An associated application for multicast is distributed shared memory, where the illusion of a shared memory is maintained in a loosely coupled network of hosts by message passing. By attaching a multicast address to each shared memory area, or even each page of memory, would allow efficient maintenance of the shared memory. Indeed, it has been postulated [De89b,Fa89a] that the advent of fibre optic networks, which offer low error rates as well as very high transfer rates, in conjunction with the shared memory architecture, will supersede the need for protocols for local area use. Other areas that use or would benefit from multicast are distributed databases [Gr78a,Pa88b], where multiple copies of a database may be used to provide greater availability or to distribute large databases across many hosts.

So far, the discussion has been about applications of multicast, and the need to address scalability because of the number of group members. Alternatively the scalability of multicast may also be affected by the environment in which the multicast takes place. One of the perennial problems in communications is congestion, where packets are discarded due to the lack of buffer space, which is used to smooth the mismatch between the rate of reception of packets and their subsequent disposal. With the increase in transfer rates achievable using modern devices, it is expected that this problem will become more significant.

The effect that network architectures have on the scalability of multicast is described in greater detail later. The main reason for pursuing this avenue is the increasing transfer rates offered by modern networks. Fibre optic based networks offer the potential for much greater transfer rates than current copper based networks, although recent developments [Fi91a] indicate that these increased transfer rates are not confined to fibre optic networks, increasing the mismatch between respective rates of packet arrival at a host and their subsequent consumption. The use of fibre optics for Local Area Networks (*LAN*s) [Li89a,Su86a], Metropolitan Area Networks (*MAN*s) [Ne88a,Ro86a] and in telephony [Mi89a] not only offer higher transfer rates but also a decrease in inter-packet intervals. When considering multicast replies, this latter point becomes significant when the synchronizing effect of a multicast is considered. Large scale multicast with replies will be developed, including the possibility of replies being transmitted with the minimum inter-packet gap exhibited by the network. The combination of the number of replies and the time taken to fully receive each packet resulting in the possibility of implosion and therefore buffer overrun. In addition, as will be described later, many protocol designs respond to packet loss by retransmission, here again resulting in possible buffer overrun. It is conceivable that in this case, a multicast takes an inordinately long time to complete with multiple retransmissions each followed by multiple replies. The effect of network and host architecture on the scalability of multicast protocols forms the major part of this dissertation, and therefore this issue is explored more fully later.

1.7 Summary

A number of multicast models were developed which included the possibility of replies to a multicast. The addition of replies allows a multicast to be reliable and transactional but introduces a potential scalability problem due to these replies imploding at the multicast originator. Implosion occurs whenever the number and rate of reception of packets exceeds the buffering capabilities of the receiving host, resulting in packets being discarded.

A multicast protocol may be characterized by a number of elements. Order describes how messages are ordered at each group member relative to messages at the other group members. Reliability describes both the guarantee an originator may assume that a message is received by group members and the number of replies that are required by an application. Scale is the subject of this dissertation and is discussed in greater detail later. A major influence on the efficiency of multicast is how messages are transferred by the underlying network, ranging from broadcast capable networks to purely unicast based networks.

A possible factor affecting scalability is expected to be the relationship between network transfer rate and the capabilities of a host to process received packets. With the transfer rates of networks increasing at a high rate due partly to the advent of fibre optic communications, this increase not being matched by similar increases for host processing, the problem of implosion is expected to become more important when designing a multicast protocol.

Chapter 2
Multicast Protocol Design

There exists an extensive body of literature concerning multicast, as illustrated by Chanson, Neufeld and Liang [Ch89a]. A large part of this literature concerns multicast protocols, which forms the main area of investigation here, as it is the protocol that provides an application with a service, that service including, hopefully, provision for scalability. However, to investigate multicast protocols it is necessary to observe where multicast is considered applicable, and what is more important here, why the scalability of multicast protocol design is considered to be an important design decision.

A number of protocol designs are described, using order as the main classification. These designs illustrate the diversity of multicast, as well as the similarities in the methods employed to achieve a particular set of characteristics.

2.1 Multicast Applications

A large part of multicast research is concerned with fault tolerant multicast, that is ensuring that messages are received by each group member in the presence of multiple failures. Fault tolerance in the context of multicast is mainly concerned with ensuring that messages are delivered reliably to surviving group members in the presence of failure. Indeed, fault tolerance may be considered one aspect of reliability. Most multicast protocol designs however assume only that reliability is guaranteed for the current group membership, so that a group member that fails before or during a multicast does not subsequently receive that multicast.

The *ISIS* toolkit [Bi89b,Jo86a,Bi87a,Ma85a,Ma88a,Bi89a] is designed around a group of hosts which are *ISIS sites*, that is run a copy of the management system. The *ISIS* design was intended to maintain a *virtual synchrony* [Bi87b] between group members in the presence of failures, assuming a fail-silent property, using well defined group management operations to maintain a consistent *group view*. All *ISIS* operations only take place between hosts that are *ISIS* sites. Each site has a group view, which describes the process groups that exist within the host group of *ISIS* sites. This group view is uniquely identified by an *incarnation number* that starts at zero and is incremented whenever the group view changes. Changes to the group view are synchronized, so that each *ISIS* site possesses the same group view at the same place relative to all *ISIS* messages. Group view changes are accomplished with a multi-phase Group Broadcast (*GBCAST*) protocol, which ensures that the group view is only changed when all operations which were in progress using the old group view have either completed or aborted. By synchronizing group changes the toolkit imposes virtual synchrony onto any processes that use the built in facilities. For virtual synchrony to be maintained requires that the other protocols used by the toolkit co-operate by using the group view for every operation. The toolkit offers three other protocols, *FIFO*, Causal Broadcast (*CBCAST*) and Atomic Broadcast (*ABCAST*). The *ISIS* toolkit is an example of a specific approach to applications which require message ordering and is included as an application as it offers a number of protocols as well as applications such as a distributed *make* [Ma90a]. These protocols are described in greater detail later.

The *ISIS* sites communicate using the reliable byte stream service offered by the *TCP* as well as the unreliable datagram service of the *UDP*. Each site maintains a connection to all other sites using *TCP*, the *ISIS* protocols using these for communication. The *UDP* protocol is mainly used in the later version of *ISIS* to allow non-*ISIS* sites to communicate with *ISIS* sites, using the *ISIS* toolkit by way of one of the *ISIS* sites.

While the above concentrated on achieving greater reliability, the use of multicast as the main communication method within distributed operating systems demonstrates a need to address scalability. Distributed operating systems are an obvious candidate for multicast, although it may be preferable to consider these as application support environments, the distribution facilities expected in such a system being used by an application proper. Any multicast protocol used within such a system must be capable of multicasting to potentially *all* of the hosts which form the operating system.

A common method for communicating in a such a system is by Remote Procedure Call [Bi84a,Co91a] (*RPC*), the paradigm often being extended for multicast use, by, for example, using a filter on replies so that only one reply is returned to the calling procedure. Examples of distributed operating systems using multicast are Amoeba [Mu90a,Ta89a], the V-Kernel [Ch85a,Ch84b,Ch83b,Ch83a,Ch88b], Meglos [Ga86a] and the Link Kernel by [Ga89a]. In addition, network computers, where the system is less highly integrated than in a true distributed operating system, also use multicast for similar tasks.

Typical uses for multicast within such systems is for communication with a distributed process group such as the sending of signals to the group, load balancing [Cr89a,Ba85b,Al87a], where the load on each host is maintained at a similar level to enhance overall performance, and other such "house-keeping" operations.

A number of applications that use the multicast support of these distributed operating systems have been described, such as a distributed *make* [Ba88a,Ba86a], using the ability the operating system to create processes remotely, allowing compilation to take place in parallel. This ability to create distributed process groups that execute in parallel is a general property of distributed operating systems. Many other examples may be described which use this facility, such as Orca [Ba87a,Baa], a distributed object manager, all of which use the multicasting facilities of the underlying operating system for process control. The usefulness of multicast for such activities is evident, and with the increasing interest in object oriented paradigms in computing as well as distributed systems, the need to address the scalability of such systems was considered well founded.

On a different note, one of the major services provided by an operating system is the file service, which with the introduction of networks may now be remotely located from file usage. The Andrew File Service [Mo86a,Ho87a] is designed to support a large group of workstations using location transparent files and file migration to enhance data availability. One of the methods used to increase the performance of file server was to allow whole file caching by workstations, the file only being written when the file was closed. One of the problems with this is the need for files being used at workstations to be updated whenever the file server copy is changed. The file server uses *callbacks*, which are registered with the file server whenever a copy of the file is requested, to transmit the file changes to each of the extant copies. For a popular file this may result in there being many copies requiring update, an obvious usage of multicast. The file server uses multiple remote procedure calls for this, *RPC2* operating in parallel [Sa90a], that is not waiting for each call to complete before beginning the next. Using multiple unicasts here may be justified by the expectation that unicast remote procedure calls formed the majority of communication. The basic Andrew File System was extended to form the Coda file server [Sa90b] which adds support for replicated files to the above basic file service thereby increasing file availability in the event of server crashes. Clients communicate with the Coda file servers using *RPC2*. Although a file service as currently conceived may be a relatively low scale system, this may change with increasing distribution of services.

A number of more general applications of multicast, such as file distribution [Co84a] and computer conferencing [Cr89b], which use proprietary protocol designs may be described. More recently, computer conferencing has been extended to include multimedia [Ro91a], which multicasts over mixed communication media such as *LAN*s and *WAN*s [Va91a], being used for passing not just text, or simple graphics, but also full motion video with the consequent real-time requirements of such activity.

In summary, multicast has an application wherever there are multiple copies of an application and messages have to be transferred to each copy. In addition, the location independence of the multicast address may also be employed where there is only one copy of an application, the property being useful for transparent application migration in case of host shutdown. Many more applications could have been described. However, the main aim here was to show that the scalability of multicast must be addressed as the group size may not be limited by some arbitrary assumption.

2.2 A Taxonomy for Multicast Protocols

Figure 1.1 showed the major design issues that are considered here to form the basic model for multicast protocol design. Of these issues, reliability, order and group management are examples of protocol design issues.

Reliability

One of the main issues in any protocol design is reliability. Reliability is often described in the context of a number of failure modes. The types of failure which may occur range from the loss of packets in transit due to noise or temporary congestion, a simple host failure, resulting in the unavailability of that group member, to a network partition, where the group members on a particular network segment are unavailable but remain operational, to the case where a member is functionally failed, but still operational and transmitting spurious or false messages, the so called Byzantine Generals problem [La82a], which is of interest because of the use of broadcast, and therefore by implication multicast, to solve the problem. The severity of a reliability failure may be measured in the time taken to clear the problem, in the case of a packet corrupted by noise, a few milliseconds, up to hours or days for a failed host. Most often, a short term failure is the realm of protocol design .

Babaoglu and Drummond [Ba85a] described how the architecture of the network may aid in solving this problem, employing multiple networks to overcome the partition problem, as well as Byzantine failure, message copies being transmitted by each network and compared, thus allowing any false messages to be discarded based on the majority view. Schneider, Gries and Schlichting [Sc84a] investigated a method for delivering messages such that the message may be guaranteed to be received by each group member, despite arbitrary failure and recovery. A different approach to the problem of ensuring reliability in the presence of failure was described by Powell and Presotto [Po83a], by using a special host to record every message on reliable storage and periodically checkpointing the group members to ensure that each member receives a copy of the message even if that member was failed at the time of transmission.

Two short term failures conditions are errors, where the packet transmitted is corrupted in some way, this corruption being detected using some form of integrity check, such as a checksum, and packet loss, caused by a lack of buffering resources at either the recipient of a packet or an intermediate host, referred to as congestion. The former may be countered by the transmission of extra error correction information either contained within the packet or as a separate check packet. The multicast models described previously indicate that it is packet loss that is the main reliability failure mechanism and it is with this packet loss that this dissertation is concerned.

The basic techniques used to achieve reliability in the design of multicast protocols are similar to those used by unicast and broadcast protocols. The protocols discussed here are considered to provide a service equivalent to the transport layer of the *ISO OSI RM*. A protocol is considered to accept a message from higher level protocols, which may be internally fragmented into packets before passing to the next lowest protocol, the maximum packet size being governed by the maximum message size of the protocol below. Each packet is then passed to the next lowest protocol, a header being attached to each packet which is used by the protocol for its own use. A packet forms a part or all of a message, the term packet being used throughout in that context. Also, a distinction between a *host group* and a *process group* must be made. A host group is a group of hosts each of which has one or more process groups resident. The *IP* multicast extensions refers to host group as typically only one instance of the *IP* protocol is resident on each host.

Three main techniques are considered. Of these, the datagram technique is the simplest, but also the least reliable, offering a "best effort" reliability. Examples of datagram protocols are the *IP* [Po81a] and *UDP* [Po80a] members of the Internet suite of protocols.

The two other techniques rely on the retransmission of missed packets or messages to increase the reliability of the transmission. In order for the originator to retransmit a missed packet the recipient must somehow inform the originator which packet was missed. Each packet is tagged using a *sequence number*, the nature of which is well understood by the protocol.

In the *negative acknowledgement* (*NACK*) technique, a small packet is returned to the originator which is generated in response to the detection of a missed packet, figure 2.1.

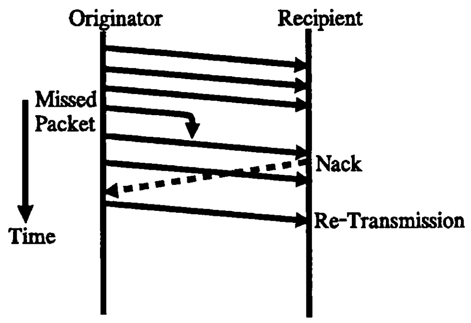


Figure 2.1 Negative Acknowledgement Technique

A missed packet is often detected by the subsequent arrival of a later packet, which allows the recipient to detect a gap in sequence numbers. The negative acknowledgement packet in effect requests the retransmission of the missed packet from the originator. The major problem with this technique is that the originator has no way of distinguishing whether a packet has been received correctly by the recipients or not, as the lack of a negative acknowledgement may mean either that the packet was received or that the recipient has not yet detected its loss. The implication is that transmitted packets must be retained until the originator is confident that the packet has been received by every recipient. Therefore, it is more correct to discuss the *degree* of reliability offered by the technique.

The last technique is *positive acknowledgement* (*PACK*), where a small packet is transmitted in response to the *reception* of a packet, a multicast resulting in an acknowedgement from each recipient. Reliability is ensured by the originator retransmitting unacknowledged packets periodically whenever there is no reply from recipients, the retransmission being governed by a timer, controlling the retransmission interval, and the number of unacknowledged retransmissions that must pass before the transmission is considered failed. Figure 2.2 shows a simple multicast with acknowledgments.

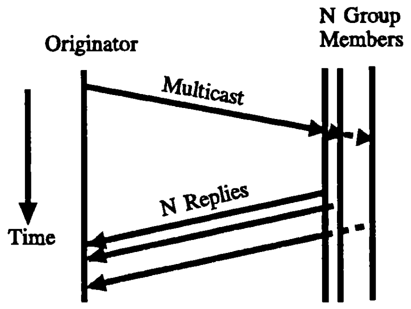


Figure 2.2 Positive Acknowledgment Technique

Again a sequence number is used, here to inform the originator of received packets. Although the use of these positive acknowledgements is considered reliable, it is reliable in the sense that the originator has confirmation of message reception. It should be noted that an acknowledgement must be received from each recipient.

Paliwoda [Pa88a] describes *n-reliable* multicast, where *n* ranges from zero to *all* in the context of transactional protocols. However, this terminology may also be applied to more general protocol designs. The value of *n* is application specific, that is the value of *n* is dictated by the requirements of the application. The service location model described previously is *1*-reliable as only one reply is required. If every reply is required then the communication is *all*-reliable. While *n* can be any value, two other values have special significance, these being *majority* and *all*, as these relate to the current group membership rather than absolute values. Using the reliability techniques described above, positive acknowledgements are all-reliable, as each recipient must acknowledge reception. Negative acknowledgements possess variable reliability using this definition, but cannot be regarded as reliable.

Congestion occurs where the number and rate at which packets arrive at a host exceeds the capabilities of the host buffering to absorb all the packets, resulting in packet loss through buffer overrun. This problem is countered by employing some form of *flow control*, controlling the number of packets generated and/or rate of packet generation, in order to reduce this buffer overrun. The simplest flow control mechanism is to transmit a packet at a time, gathering all acknowledgements before transmitting the next packet. If the round trip delay of a network is small, then the overall delay involved is also relatively small. However, if the round trip time is large then this method is inefficient. By allowing a number of packets to be transmitted in a block, an acknowledgements being transmitted at the end of a block, then the overall delay experienced *per packet* is reduced. The number of packets transmitted may be controlled by using a token that gives an originator permission to transmit a certain number of packets, the value of this token being based on the amount of unused buffer space at the recipients. The token, called a *window*, is most often associated with the positive acknowledgement above, the acknowledgement packet also containing the window information. Indeed, most implementations of a window system require reply traffic in order to operate. This type of flow control is extensively used in unicast communication, an example being the Transmission Control Protocol (*TCP*) [Po81b,Cl82a,Ja90a]. Studies into the efficiency of windowing schemes for multicast over broadcast satellite are described by [Go84a,Ra86a] and [Mo87a].

An alternative is to control the rate at which packets are generated, by interposing a timed gap between packets at the originator. The recipient may change the rate at which packets are transmitted by informing the originator either that packets are being missed, or by specifying a preferred transmission rate. An example of a protocol employing rate control is the *NETBLT* protocol [Cl87a,Cl87b]. These flow control methods are equally valid for multicast as for unicast, and indeed many protocol designs use them. However, because there are now many recipients, each of which apply flow control, some means is required to prevent some recipients from altering the overall flow control such that other recipients suffer congestion. The main requirement here is that flow control information be received from each recipient before any change is made. While these methods control the flow from the originator, controlling the flow from recipients to the originator is more difficult, especially if these reply packets are not traditionally subject to flow control, an example being acknowledgements.

When discussing multicast protocols two main measures are often used to compare designs. These are the *completion time*, that is the time delay between the transmission of the first packet of a multicast and the reception of the last expected reply, and the packet efficiency already described. One of the primary aims of many multicast designs is to reduce the total number of packets transmitted per message to a minimum, subject to reliability and underlying network architecture constraints. The main factor in the packet counts exhibited by a protocol is the technique used for data transfer, the packet counts exhibited being modified by the two constraints introduced above.

Group Management

Group management is defined by group management operations, which operate on what may be termed *management information.* The structure employed for this information is influenced by the frequency at which the various operations take place. Simple well known applications, where there is a permanent mapping between the service and its multicast address, have no need for group management, the existence of the group being tested by simply multicasting to the group. However, the majority of multicast applications would be expected to *create* a multicast group for themselves, using an unused multicast address, the first application being responsible for group creation. Two problems are immediately apparent; firstly how to discover an unused address, secondly how to inform joining group members and users of the group which address maps to which group or service.

A method much used for the discovery of an unused address is to multicast to an arbitrarily chosen address and if replies are received then to attempt another. The efficiency of such a scheme is reliant on the use of a sparsely populated address space to ensure that the number of address clashes is small. Hughes [Hu88a] describes an algorithmic approach to address generation, by taking the host address and appending the current time of day to form a unique multicast address. Cheriton [Ch88b] employed an address *manager* for the management of process addresses in the V-Kernel which by implication could be used for the same role for multicast addresses. A feature of this implementation was that each host in the operating system requesting a block of addresses at a time to reduce the number of times that the service was called. The *ISIS* system allocates unique addresses because of the synchronous nature of group management operations, so that the group view at each site is identical. The mapping of addresses to application groups is not well documented, the mapping being implicitly assumed.

The converse of group creation is the *destroy* operation, where the address is deallocated and returned to the pool of unused addresses. The commonest method for this is for the group to be implicitly destroyed when the last group member leaves the group. While this works well when addresses are discovered using the simple approach described above, if the address was allocated the address manager must either monitor the address for activity or be actively informed of the group's destruction. It should be noted that any users of the group must also be made aware of the group's destruction, which may be achieved by simply delaying address re-use allowing the user time to detect the destruction of the group. Ngoh and Hopkins [Nga] describe the use of an explicit destroy operation, in this case occurring when the creating member leaves the group regardless of the current group membership, forcing any remaining members of the group to leave the group.

Once a group is created then applications may *join* and *leave* the group. Because the group address is already allocated, a joining application must have some means of discovering the correct address to use. In the stricter management regimes this is accomplished using the group view to provide the mapping between application and address. The equivalent leave operation is mainly used to inform the group management of the fact to enable the group membership to be updated, but may also be used to delay the actual departure of an application until any extant data has been received.

Two other operations are also defined, these being *failure detection* and *query*. The detection of failure is important if the group management must be accurate for the correct operation of the system, the main method used being for a group leave to be issued on behalf of the failed member. The query operation is used mainly by non-group members to allow access to the group, and by group members to assess the current group membership. Figure 2.3 shows how the group management operations relate to each other.

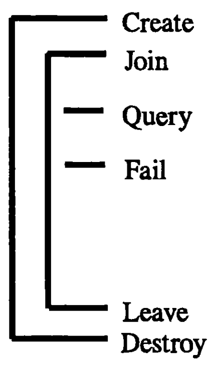


Figure 2.3 Structure of Group Management Operations

The structure of management information may range from a fully centralized system where all operations act in a single place, through a group centred structure, where the operations are centralized on a per group basis to fully distributed where either full or partial information is replicated at each group member. The information itself may simply be a mapping between multicast address and the application name to a list of all of the group members currently active, with each entry possessing specific information about the indicated member and is often termed the *group view.* In a dynamic group environment, where applications are joining and leaving the group, the timeliness of updates to the management information must also be addressed, as well as the transfer of this information to joining members, which in addition has implications for order described below, specifically whether group management changes should be ordered relative to data transfers so that the data transfer is directed to the all of the current group members.

Order

The importance of order in multicast may be seen from the number of protocols which are designed with order as the prime design goal. One of the problems with multicast that is a consequence of the grouping nature of the method is *consistency.* The N:M model of figure 1.1 shows that with multiple originators and multiple recipients that message may be received in a different order at each recipient. For data replication, where multiple copies of data are stored at a number of hosts, the problem is to maintain the consistency of this replicated data to ensure at least that the most recent version of the data may be accessible to any application. *Voting* may be used, where data is tagged with a vote, the application having to gather a quorum of votes so that at least one of the data copies is the most recent, the data having been written in such a way to ensure this. Voting is a popular method of controlling replication, as evidenced by the work by Herlihy [He86a] Barbara and Garcia-Molina [Ba86b] Long et. al. [Lo88a,Lo88c,Lo88b,Lo90a], Gifford [Gi81a] and Kumar [Ku91a].

Multicast ordering is related to those used for voting by being intended to maintain the consistency of data replicated at many group members. Multicast ordering is more concerned with the order in which messages are processed at each recipient relative to the other group members, so doing ensuring that replicated data is maintained in a consistent state. The difference here is that multicast ordering is concerned with maintaining the consistency of data at the group members, rather than ensuring that any application is able to access the most recent copy of that data, so that *all* copies held by group members are the most recent. The role of order in distributed systems is explored more fully by Birman and Marzullo [Bi89a].

Multicast order becomes a problem whenever many originators communicate with a group. If there is only one originator then order is automatically guaranteed if the originator transmits messages one at a time, so that each multicast is completed before the next begins. If there is only one group member, then again there is no problem with order, as all messages are then processed in the order of arrival. Indeed, this fact forms the basis for a number of protocol designs described below. A distinction must be made between the *reception* of a message at a recipient and its subsequent *delivery*, many ordering protocols increasing the delay between reception and delivery to ensure the ordering paradigm, which in turn increases the *latency* exhibited by a protocol, which may be defined as the delay between message transmission and final delivery of the last copy of that message. Latency is also affected by the completion time of a protocol, as order cannot be guaranteed until all of the messages that are subject to that ordering paradigm are received by each recipient, which implies that the transfer time between originator and recipients is of importance. A longer transfer delay also increases the probability that further messages are transmitted which need to be ordered. Indeed, it is partly because of this delay that ordering is required. If the transfer delay is zero, then messages are automatically ordered by the time of transmission, which implies that the longer this delay and greater the dispersion, the greater the disorder. This is one of the reasons why many of the ordering protocols described later are confined to networks which exhibit low delay and dispersion.

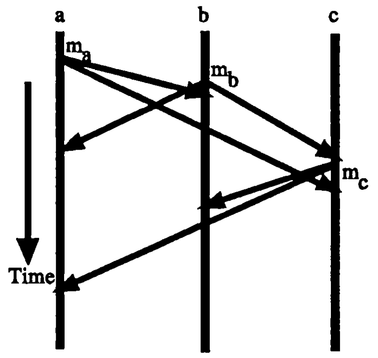


Figure 2.4 Arbitrary Order

Definitions of order depend on the type of ordering paradigm, but can loosely be described as the relationship that any particular message has with other messages in a distributed system. The relationship is generally defined by four types. **Arbitrary order**, figure 2.4, is the simplest ordering property and describes messages ordered by time of arrival independently of that message's order anywhere else in the system and is the basic service offered by all multicast (and unicast) protocols. The only restriction imposed is that messages transmitted from the same application are processed in the order that they were sent, which is easily achieved by using a sequence number in each message. From figure 2.4, host *a* processes message *a*, then *b* then *c*, whereas host *b* processes message *b* first, then *a* then *c*.

A stronger ordering type is **group order** that ensures that messages are delivered to each group member in a consistent order, that is each group member is guaranteed to receive messages in the same order as every other.

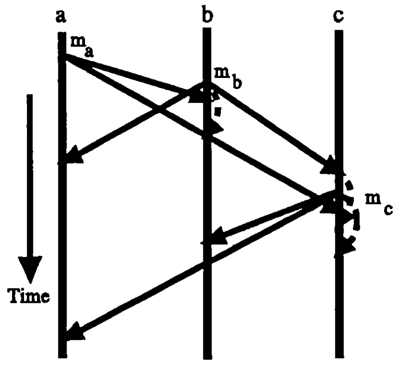


Figure 2.5 Group Order

This is often referred to as *serializability*. It should be noted that the order of messages is arbitrary, that is there is no guarantee that the order reflects a desired sequence of messages. Figure 2.5 shows how the delivery messages are delayed at hosts *b* and *c* so that the same order is established at each host, here a before b before c.

Extending group order to multiple groups results in **group overlap order** where, not only are messages guaranteed to be ordered identically at each group member, but also all messages at each group member which is also a member of other overlapped groups. Groups overlap if different group members are resident on the same host, with messages destined for each group ordered the same at every host that exhibits this overlap.

The last ordering paradigm considered is **desired order**, where messages are delivered in the order *requested* by the originator. Desired order is a partial order based on events, such as the reception or transmission of a message, that are said to have occurred *before* the current event, so that a message that is causally dependent on another message requires that this later message has been, or will be, received by each group member that receives the former message before the former message is processed. Lamport [La78a] defines a causal relation by the symbol ->, so if *a* occurs before *b* then a->b. Figure 2.6 shows how causality may be employed, the dotted slanted line from host *b* to host *c* represented the transmission of message *a* to *c* by host *b before* message *b*, message *b* being causally related to message *b* and therefore required at host *c*, which was not originally a destination of message *a*.

A property often associated with order is *atomicity* which is an *All or Nothing* paradigm, stating that either all group members receive a copy of the message, which can subsequently be processed, or, in the event that some do not, none are allowed to process the message, that is the message is discarded before delivery. Atomicity can either be originator based, that is if an originator fails during transmission then the message is aborted, even if some of the recipients have received it correctly, or recipient based, where the message is aborted if any of the recipients fail during transmission, with the message being resent by the originator when the group membership is stable again.

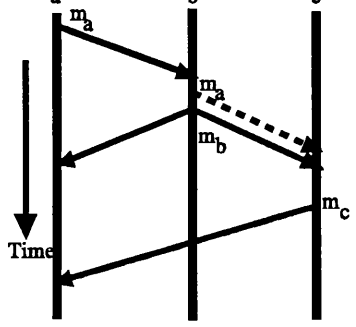


Figure 2.6 Desired Order

2.3 Arbitrarily Ordered Protocols

The protocol by Erramilli [Er87a] was designed for use on a broadcast *LAN* and sit at the data link level. The protocol uses a *back buffer* at each originator to store transmitted packets, packets being pushed out of a full buffer and destroyed whenever a more recent packet is put into it. One of the problems with the negative acknowledgement technique is the detection of a missed last packet, as there is no subsequent packet to prompt the protocol. This design uses a timer, which is set by the originator on the transmission of every packet. If this timer expires, as would occur when there are no more packets to send, a *status* packet is transmitted bearing the sequence number of the next packet that will be transmitted. The status packet is transmitted periodically a number of times, giving the recipients several opportunities to receive the status packet. Of course, if a data packet becomes available for transmission during this period the timer is reset and the sending of status packets halted. The status packet has a sequence number one greater than the last data packet, which allows the recipient to detect any missing packets, acting both as a reliability mechanism and a lost last packet detector. The status packet is superseded by the *sanity* packet, which is transmitted periodically, the interval being greater than for the status packet, and is used to confirm the presence of a recipient.

The main interest of this protocol is the analysis of the scalability of the protocol in comparison with a positively acknowledged design, although the scalability is modest. The analysis compares two activities of the protocol, namely the *lecture* mode, where one originator is transmitting the broadcast group, the other, *conference* mode, where packets are transmitted arbitrarily by every host. It should be noted that the negative acknowledgement technique is extensively used by many of the ordering protocols described below, the same issues having to be dealt with for these as above.

An innovative protocol design, mainly used for unicast but with multicast capability, is the Versatile Message Transaction Protocol (*VMTP*) [Ch89b,Ch86a] which was originally designed for use by the *V-Kernel* described above as the main Inter-Process Communications (*RPC*) mechanism. The protocol has since been offered as a transport level Internet protocol [Ch89b,Ch88a]. The protocol's design is transaction oriented, with additional emphasis on performance with large amounts of data. The protocol is of interest mainly because of the selective retransmission policy and the rate control employed as a flow control mechanism. The protocol assumes a standard, fixed maximum transmission unit of 512 bytes of data, to which is pre-pended a relatively large header. The protocol can handle up to 32 of these chunks at a time, enabling up to 16 Kbytes of data to be transmitted as a chunk. Each packet in this chunk is represented by a bit in a 32-bit long field of the header. An acknowledgement sets the respective bits in its own field, forming an *acknowledgement mask*, representing the packets it received. The originator, on receiving this acknowledgement, can observe any patterns of missed packets and adjust the rate at which packets are transmitted, as well as retransmitting the indicated missed packets. Rate control is used to prevent an originator transmitting packets faster than recipients can receive them. The transmission rate is controlled by the protocol "busy-waiting", that is executing a null loop to delay the protocol execution. While this is inefficient, it provides the small time intervals required which may be too small for the operating system's internal clock to handle. One feature of the protocol is that the transmission rate is dictated by the recipients, each recipient informing the originator of the preferred rate, which can be subsequently manipulated using the acknowledgement mask described above.

The protocol uses large 64-bit identifiers as addresses for both unicast and multicast transmissions, multicast being indicated with a portion of the address being allocated for group operations, which are manipulated by a set of group management operations performed by a *management server* that is co-resident with each *VMTP* host implementation. The create operation requests a block of addresses from the management server, to reduce the number of queries to the server.

Beyond this there is little use of management operations, as group members join and leave groups with impunity. The philosophy used in the design of the *VMTP* is that the design is 1-reliable, that is only the first reply is significant. While 1-reliable transactional multicast is suitable for service location and the querying of idempotent services, it does not cover all cases.

The protocol by Crowcroft [Cr88a] is also a transaction protocol, with similar properties to *VMTP* but using a sliding window for flow control, which implies a *go-back n* type of retransmission strategy. The protocol is stream oriented, and is similar in many aspects to the *TCP*. Because each recipient will reply with a potentially different window size, indicating the number of packets that each recipient can accept, the returned windows are *coupled*, by choosing the lowest offered window, ensuring that the most congested recipient controls the transmission. It is conceivable that a different strategy could be used, especially bearing in mind the use of *optimistic* window sizes, where the offered window is greater than the available buffer space. As well as coupling the windows, the operation of the retransmission timer is also coupled, to the recipient with the longest round trip delay.

The design offered an *n-reliable* service, with *n* ranging from zero to the current group size. The protocol itself was required to be *at least n-reliable*, that is at least n replies to the multicast must be returned, although only *n* replies are passed to the application, other messages being discarded. The design provided a means of supplying a filter function in order to discard messages at the protocol level rather than passing unwanted messages to the application.

Because the protocol required replies, the design was perceived to be affected by the *implosion* problem. Therefore, to disperse the transmissions of a reply over time a small random delay was introduced by each recipient to delay transmission. The paper investigated the effect of using small additional delays on an Ethernet, showing that there exists an optimum range for the inter-packet gap which reduced collisions, and therefore the completion time of the protocol.

Group management for this design was assumed to exist external to the protocol, the protocol acquiring a list of group members from this management service. The protocol used this list as a check when gathering replies, with a retransmission occurring if no reply was received from a "known" host. The operation is similar to the use of a connection in the *TCP*, except that the connection is made implicitly from the supplied address list, rather than from a packet interchange.

The design by Danzig [Da89a] is of interest because the protocol is designed for multiple retransmissions, directly reflecting the perception that implosion has a significant effect on the design of a protocol. The intention was to reduce the overall completion time of the protocol with a combination of an optimally based delay algorithm, to reduce missed transmission opportunities.

Because packets were *expected* to be missed the protocol ensures that as a message is retransmitted the set of recipients which reply to the multicast is reduced because the originator has already received an acknowledgement from them by acknowledging the reception of replies in the retransmission, using a bitmask in the header, one bit per recipient. Therefore, after a number of *rounds* the number of replying recipients goes to zero and the multicast completes. The *round time* dictates the interval between transmissions which are uniformly distributed. By reducing the number of replies with repeated retransmission the protocol is an example of the need to consider scale as a design issue.

The delay algorithm is similar in function to the one described above, but is deterministic rather than random. The calculation of the delay time was made to minimize the cost of a multicast, in terms of originator transmissions, recipient responses and the completion time. Each recipient uses its own response time to a multicast and the controlling parameters of the multicast. The delay time is divided into slots, so that each recipient has a slot in which it can reply, the reply being delayed until the next round if the slot is missed. The allocation of these slots by each recipient is controlled by the optimal distribution, which is calculated using the values described above. It must be stated that the use of an optimal delay scheme on a network that does not globally support such a scheme may not exhibit the optimal characteristics in the presence of external traffic.

Group management is, again, assumed to exist. The management operations are complicated by the need for each recipient to know which bit in the retransmission bitmask represents them, joining applications having to have a free bit allocated to them. The use of a bitmask in this way also potentially limits the maximum group size to the number of bits in the mask. The need by a recipient to discover a "free slot" extends to the optimal delay strategy as well, as the most optimal strategy is one in which messages are transmitted in such a way that the inter-arrival time of messages equals the ability of the host to consume them.

The protocol by Errimilli is not reliable, although the degree of reliability is enhanced by the transmission of status packets after a packet to allow each recipient additional time to detect a missed packet. No flow control is employed. The other protocols in this section are all positively acknowledged with the multicast being reliable. However, the *VMTP* is only guaranteed to be 1-reliable for replies, the philosophy underlying the design since only the first reply is significant. Flow control is employed by each design, exhibiting three methods for this, namely a window, rate control and the retransmission of packets with an effective reduction in replies for each round. These protocols are not fault tolerant, as there is no provision for the recording and subsequent retransmission of messages missed by failed hosts. The packet efficiency of the negatively acknowledged design is high, as it can be expected that few negative acknowledgements will be transmitted given the low loss characteristics of a *LAN*. The positively acknowledged designs however show a steadily decreasing packet efficiency as the group size increases.

2.4 Group Ordered Protocols

The Atomic Multicast Protocol (*AMp*) by Verissimo et. al. [Ve89a] is based on a two phase approach, figure 2.7, and was designed for use on token ring networks.

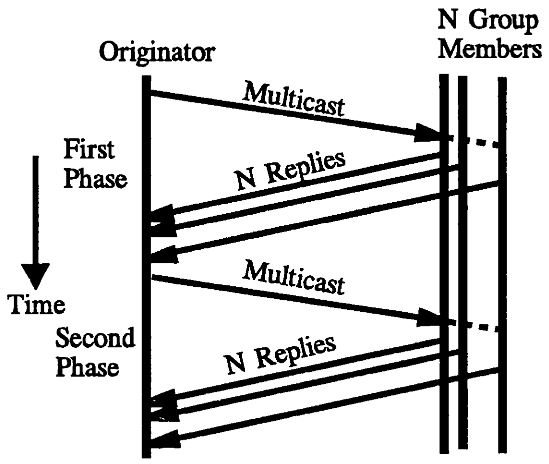


Figure 2.7 Two-Phase Pesign

The use of a two-phase design is common in systems which exhibit atomicity, the first, *distribution*, phase being used to transmit the data, the second, *confirmation*, phase to confirm or abort the delivery of that data. The protocol is implemented at the link layer and is designed to be decoupled from the host and therefore resides in the Network Attachment Controller (*NAC*). These *NAC*s are considered to be fail-silent, that is on failure they do not transmit any more messages. The network environment is a dual token ring for additional fault tolerance.

Each protocol instance is built from the *Emitter Machine*, one or more *Receiver Machines* and a local *Group Monitor* agent, which aids in error recovery and fault detection and recovery. The protocol uses two context structures, a *Group View* and the *Receive Queue*, containing the current group composition and references to received frames respectively. The group view has two aspects, a *concise* and an *extended* view. The former is just the number of group members currently recognized, the latter is a full list of the actual group member unicast addresses. The use of a concise group view may be supported because of the packet orientation of the protocol, acknowledgements being generated for each packet.

The distribution phase of the two phase protocol is initiated by the transmission of a packet to the multicast group, and ends when either all the intended group members, identified from the *group view* have replied with either an *accept* or *reject* acknowledgement, the latter being returned if a recipient cannot buffer the packet, or if a preset timer expires, which allows for a number of retransmissions of the packet. The decision phase then informs the group of the originator's decision. Serial order is guaranteed by utilizing the exclusivity property of the network, that is because only one packet can be traversing the ring at any time, then if all of the intended recipients receive that packet, order is maintained relative to other packets. In the event of a packet loss then the packet is retransmitted, *any older copies of that packet that were previously received being removed from the respective receive queues*. In effect the packet loses its place in the receive queue.

The *Group Monitor* function is used to maintain the coherence of the group views and to execute a termination protocol in the event of an originator failure. The termination protocol transmits an abort message to the group if an originator fails at any point in the two phase multicast. The group monitor function is executed by an *Active Monitor*, at most one of which exists per group. The Active Monitor is elected, *when required*, from a number of *Inactive Monitors*, the Active Monitor existing until usurped by the election of a new Active Monitor. Such a scheme is recursive to counter Active Monitor failures. Contention and deadlock are resolved by using a *Suspension* attribute which suspends multicast activity for the group, forcing the other Group Monitors to enter a standby state. If more than one Group Monitor transmits a suspend message, then the contention is resolved by the use of a *suspension level*. Deadlock is resolved using a timer which is reset whenever a message from the Active Monitor is received, the expiry of which assumes the failure of the Active Monitor, resulting in the execution of the election algorithm. The group view is subject to three group management operations, join, leave and failure.

Another example of the two phase approach to serializability is the *Atomic Broadcast* (*ABCAST*) protocol used in the *ISIS* toolkit [Bi87a,Jo88a].

The original *ABCAST* protocol used a timestamping approach to serializability, the timestamp being returned by the recipients as part of the dissimation phase, each recipient placing the newly received message on a list of pending messages and marking it as undeliverable. Each group has a separate queue for these pending messages. The timestamp is chosen to be larger than any other timestamp assigned or received by the recipient in the past. Timestamps are guaranteed unique by appending a recipient specific number to it. The originator collects these timestamps, chooses the largest value and transmits this value to the recipients. This is the *final timestamp* for that message. Each recipient on reception of this final timestamp assigns the value to the message, marks the message as deliverable and then reorders the queue of messages by increasing value. If, after reordering, the first message on the queue is deliverable, the message is removed from the queue and delivered to the application, this process continuing until either the queue is empty or the next message is marked as undeliverable.

While the protocol is described as being atomic, the atomicity is biased towards the failure of the originator, rather than the recipients. The failure of a recipient is ignored by the protocol, but the failure of the originator requires that one of the recipients that have already received the message ensures that the protocol completes.

A later version of the *ABCAST* protocol, while observing the same semantics, uses a variation on the *CBCAST* protocol described above. It relies on the fact that multicasts from an originator are automatically ordered by the time of transmission as long as a multicast is fully complete before any other originator is allowed to multicast. To use this method some means of identifying a single originator is required. A token is used for this purpose.

The token holder has exclusive right to perform an *ABCAST* thereby ensuring the serializability of messages. Other originators have to request the token from the token holder before they in turn can transmit. The failure of a token holder results in the execution of a distributed algorithm by each group member to create a new token. If a requester fails before the token is handed to it, then the resulting inability to transfer the token results in the next requester being tried. The paper by Birman and Marzullo [Bi89a] describes a method of using the *CBCAST* protocol to maintain a distributed list of requesting originators at each site. The list is group ordered to the other lists, but because it is the token holder's task to choose the next token holder, and therefore the next originator, then the only requirement is for the recipient of the token to be deleted from all of the lists. Such a token passing scheme would benefit only those applications which exhibited *locality of reference*, that is an originator transmits many multicasts, before the token is requested by another group member. By using a list of requesters, it can be assured that only one request from each originator can be pending at any time, so that fairness is guaranteed.

A variation on the *ABCAST* protocol is used for maintaining the *group views* at each *ISIS* site. Because the group view is used by all of the protocols which comprise the *ISIS* communications suite, the update of the group view requires a protocol which is ordered relative to all of these other protocols. However, the *Group Broadcast*

(*GBCAST*) protocol is not a global ordering protocol. It operates only on the group members of a particular group, not all groups. *GBCAST* uses at least three phases to ensure that the group change is carried out in the same order relative to any messages that have not yet been delivered to the group as a whole. The group management employed by the above protocols revolve around the group view and the use of *GBCAST* to maintain the consistency of that view.

A *GBCAST* is invoked whenever any of the closed group management operations are used, the group view being propagated, to not only group member sites, but also to any site that expresses an interest in that group. A site failure is countered by the detecting site issuing a leave operation by proxy for the failed site, so that a failure and an intentional leave are transparently integrated by the leave operation.

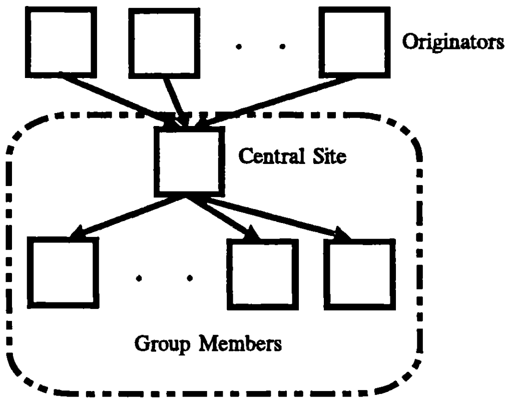


Figure 2.8 Single Site Ordering

The use of a token to enforce a single originator policy, that is restricting multicasts so that only one originator multicasts at any time, is a variation on the use of a *single* or *central* site for serializing multicasts. The single site acts as an intermediary between the originators and the intended recipient group, figure 2.8, and is most often a member of that group. The single site approach uses the fact that order is guaranteed if there is only one group member or there is only one originator of multicasts. By combining these, designating one of the group members as both the recipient of all messages to the group, and the originator of all multicasts, order may be guaranteed. The *ABCAST* protocol described above is an example of this type of method.

The single site method may be described by a series of transmissions. The originators freely transmit in such a way that the single site receives the message, that is there is no restriction on when originators can transmit.

The single site then acknowledges the message, so that the originators have confirmation that the message has been received, at least by the single site. The single site then transmits the messages in the order in which it originally received them, the recipients acknowledging the message reception. Although the above describes the steps taken, the number of messages transmitted is quite high compared to what may be achieved, as described below.

The protocol by Navaratnam, Chanson and Neufeld [Na88a] uses *managers* for all communication. Each group is represented by one *primary* manager and zero or more *secondary* managers, with one manager per host. Each manager keeps state information about the processes local to the manager, as well as the locations of all other managers in the group, reducing the overall amount of state information held by any host. Two multicast protocols form the message transfer mechanisms, *UGSEND*, a *FIFO* ordered multicast which may be used arbitrarily by any process, the message being transferred directly to the group members, and *OGSEND*, which is serializable.

An originator invoking an *OGSEND* transmits its message to the primary manager, which acknowledges the message. The primary manager then transmits the message to each of the secondary managers of the group, as well as passing copies of the message to its own processes. Each of the secondary managers acknowledges the message and passes it to their own processes. To improve the transfer rate of messages, the primary manager assigns a sequence number to each message before transmitted, so that the secondary managers can reconstruct the transmission order based on these sequence numbers, rather than on time of arrival, allowing message to be transmitted without requiring that the previous message be acknowledged. The actual transmission can be by multiple unicast or a multicast, the transmission being reliable.

One of the problems with the single site approach to order is that of the failure of the single site, which would require that another site take over the functions of the single site. The problem is solved for the above protocol by using a *vulture* process to monitor the primary manager at each secondary manager. When the failure of a primary manager is detected by this vulture process then an algorithm is used to elect one of the secondary managers to be the new primary manager. The new primary manager is responsible for checking that each of the secondary managers is in the same state, as it is possible that the primary manager failed during a multicast.

The primary manager is the focus for all group management operations, as it is the primary manager's job to construct a view of the current secondary managers and propagate that view to each of the secondary managers. The managers are responsible for forwarding received messages to each resident application.

All of these designs are reliable, indeed all-reliable, as the originator must have confirmation that every packet has been received by the group in order to make a decision about the order in which it is to be placed. The two phase approach requires two rounds of this, which results in a poor packet efficiency and a long delay relative to the single site approach.

The Multicast Transport Protocol by [Ar92a] uses a similar scheme to that described above, except that instead of the message to be ordered being transmitted to the primary manager, a token is requested from the group *master*, which contains the sequence number to use for the transfer, reducing the amount of information being transmitted in total. The master is also used for the group management join operation, arbitrating the join process and assigning control values to the joining application protocol.

Data transfer uses a negative acknowledgement scheme controlled by a window, which dictates the maximum number of packets that will be transmitted in any block. The packets may be of two types, data packets and *dally* packets. By always transmitting *window* packets, window being considered as a number here, the probability of at least one of the packets being received is increased, so that missed packets may be more easily recovered from the originator. A *heartbeat* value was used as the basic timing feature, one or more packets being transmitted in each heartbeat interval by the protocol.

2.5 Overlapped Group Ordering Protocols

The protocol by Kaashoek [Ka89a] is a single site protocol which uses broadcast for packet dissimation and is therefore only usable on a broadcast network. In addition to providing a monotonically increasing sequence number to each received packet, the *sequencer* has the responsibility of retaining a number of previously transmitted packets in a *history buffer* so that any host that missed a packet can use a negative acknowledgement to recover it. An originator transmits a packet to the sequencer as a unicast. On reception the packet is assigned a sequence number and broadcast, a copy of the message being added to the history buffer. The protocol reduces the number of packets transmitted by using the transmission of the newly sequenced packet as its own acknowledgement, the packet being broadcast and therefore, by definition, received by every host. If the originator does not observe this broadcast within a preset interval, then the packet is retransmitted. The history buffer has a fixed maximum size and so the sequencer uses state information about each host to determine the lowest sequence number seen by the hosts, enabling the sequencer to discard all cached packets with a lower sequence number. The state information is transmitted to the sequencer in the data packets or, periodically using a *NULL* packet. In the event of the history buffer becoming full the sequencer enters a two-phase synchronization period, where no packets are accepted and the hosts are prevented from sending any new packets.

The history buffer is flushed by first informing all the broadcast group members that the sequencer is discarded all packets older than a certain sequence number. Any host that does not have one of the indicated packets uses negative acknowledgements to recover the missed packet. When each host has indicated that it has all the packets that are to be discarded, the sequencer removes them from the history buffer and then issues a commit packet, indicating resumption of normal service, which must be acknowledged by each host before the sequencer accepts new packets.

The failure of the sequencer site and the location of the sequencer by originators is not dealt with by Kaashoek, for the protocol to be fault tolerant an alternative sequencer site must not only take over the functions of the failed site but also reconstruct the state of the failed sequencer so that no packets may be missed by negative acknowledgements. Also, the new sequencer's address must be communicated to the originators.

Because the protocol is based on broadcast there is no direct use for group management, as by default all hosts are group members. The sequencer is required to maintain state about the hosts but only because of the method used to reduce the size of the history buffer, and is not considered to be relevant to group management. The protocol may be modified by using an *all-multicast* address to reduce the number of wasted packets, with packets being directed only to those hosts that have a multicast group resident. The design is reliable to the originator, as the original packet is acknowledged by the reception of the subsequent broadcast. Flow control is based on originator retransmission if the packet is not acknowledged, the recipients using negative acknowledgements if they miss packets.

A variation on the single site approach is by Chang and Maxemchuk [Ch84a]. The variation is in the fact that the sequencer is rotated among the group members, the current sequencer site being indicated by a token. The protocol was designed for use over a broadcast satellite system so all packets are broadcast as above. Indeed, apart from the use of a circulating token the two protocols are essentially the same.

The main difference between the two is that the initial data packet is also broadcast, so that it can be received by every host at that time. The acknowledgement is broadcast by the token holder, which offers a second chance for the sites to get a copy of the packet if it was missed on the original broadcast, using a negative acknowledgement. All negative acknowledgements are directed to the token holder, not the originator.

The token system forms a ring, with the token being passed from host to host by an acknowledged broadcast. A reduction in overall traffic is achieved by piggy-backing the passing of the token onto a data packet acknowledgement. The ring structure is enforced by each host retaining information about the hosts to which it passes the token. This *token list* is maintained by a *ring reformation* protocol, which is invoked whenever a host fails, recovers or hosts are using different token lists to the one expected. To ensure that a token holder can satisfy requests for missed packets, the token can only be passed if the designated next token holder has a copy of each of the packets that remain in circulation. If this is not the case then all the missing packets are recovered from the token holder which, by definition, must have a copy of the missing messages.

The protocol delays the passing of packets to applications until a *resilience* test is passed, which refers to the number of hosts that can be guaranteed to possess a copy of that packet. The passing of the token is used to guarantee that a host has a copy of each packet, therefore the passing of the token N times ensures that *at least* $N + 1$ hosts have a copy, after which the message can be committed. Further, after the token has traversed the ring once, each host can be guaranteed that the packet has been committed by every operating host, and therefore the packet can be removed as the token circulates a second time.

A reformation phase is used when the ring structure is broken, either by a failure or the recovery of a site. Again, because the protocol is broadcast, there is no group management as such. The reformation protocol is a three phase protocol that ensures that a new token list is received by each of the group members, elects a new token holder and that none of the committed messages from the old token list are lost.

The protocol by Garcia-Molina et. al. [Ga88b] uses a hierarchical method of ordering messages across all groups by building a tree structure from the groups and group members using a Propagation Graph (*PG*) as a routing path for multicast packets, the graph being generated by a *PG* generator. Each host forms a node of the graph, with messages being passed from node to node, following the path dictated by the *PG* using the Message Passing (*MP*) protocol. The *PG* generator uses the number of groups at nodes as a means of reducing the traffic at each host at the same time ensuring that there is only one path followed by messages, so that messages are ordered as they traverse the tree. Originators transmit to the *primary destination*, which is the member of the destination group closest to the root of the tree. Figure 2.9 shows a number of such trees.

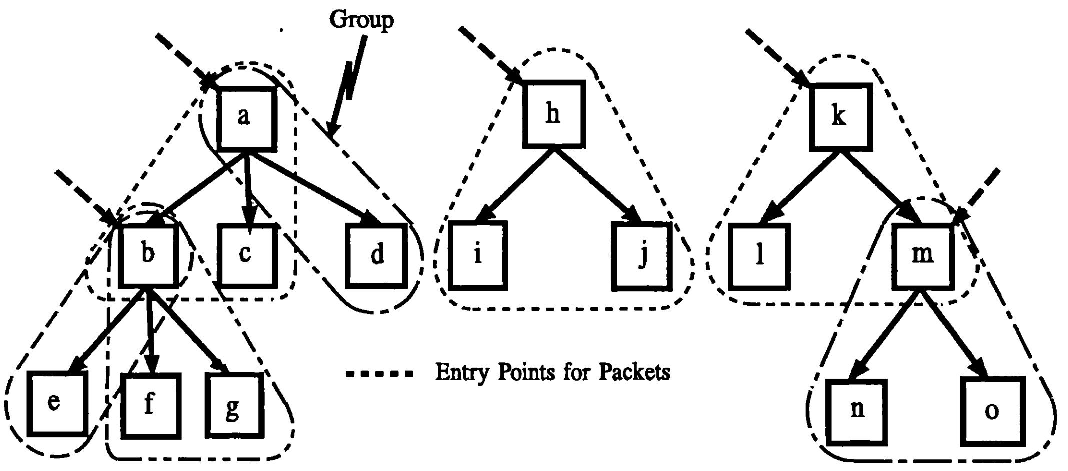


Figure 2.9 Multicast Trees Generated by *PG*

The *PG* calculated whenever group management operations occur which change the state of the tree. It was assumed that one host would calculate the *PG* and transmit it to each of the other hosts which belong to the *PG*. The tree is built be gathering information about the number of groups resident on hosts, the *PG* beginning with the host which has the largest number of groups resident. A recursive algorithm is then called which partitions all of the groups not in the *root groups* in such a way to ensure that no group in a partition intersects a group in another partition. An intersection occurs when two groups are resident on the same host. The algorithm is then used on each partition using the same starting point as above. The algorithm continues until all the partitions are singletons. In the event that some groups have no intersections in the first place, then the *PG* generator is executed for each of these, resulting in many trees, a *forest* of propagation graphs.

The *MP* protocol is used to forward messages from the primary destination to all of hosts which are children of the primary. Sequence numbers are used to maintain consistency, each *MP* protocol maintaining two queues, one for local messages and one for messages that arrive out of sequence. By delaying the forwarding of out-of-sequence messages, the ordering guarantee can be maintained.

The performance of the *PG* technique depends on the depth of the trees generated and the number of extra nodes required for intermediate routing. The latter is of concern when the technique is used over a *WAN* such as the Internet, where extra nodes are used for routing purposes only. When the *PG* technique is used in a broadcast environment there are no extra nodes, so that the cost is just the depth of the tree plus the initial transmission to the primary destination.

The protocol can be used in a dynamic group environment by appointing one of the hosts as a manager that has the responsibility of computing a new propagation tree and causing the new tree to be passed to all of the hosts. The old graph is *closed* by using the old *PG* to send a close instruction to each root, ensuring that the close is delivered in order. The close ensures that the *MP* protocol stops forwarding using the graph, but does not stop originators from transmitting to primary destinations, these messages being queued. When a new graph has been calculated, it is forwarded to the hosts using the new graph. The new graph is *opened* using an open instruction, which informs primary destinations that they can continue operations, and any other host that has messages has to throw them away and request the originator to retransmit them to the new primary destination for that group. Because the close is propagated using the old graph, it is guaranteed that all messages transmitted before the close have been delivered, and any after were queued at the old primary destination, so it is only those hosts that were primary destinations, and are no longer, have to throw away messages. This assumes that the originators have some form of history buffer to recover messages.

All of the above protocols guarantee that packets will be ordered properly. An example of an optimistic ordering protocol, which offers only a high probability that all messages will be ordered at each node identically, is by Melliar-Smith [Me90a]. In order to guarantee global ordering the *partial order* achieved using the optimistic protocol is manipulated by a second protocol. The *Trans* protocol achieves a partial order using an *Observable Predicate of Delivery* for reliability, to determine which hosts have received a message, even if they have not directly acknowledged the message.

The *Total* protocol constructs a fully ordered system from the partial orders constructed by each host.

The *Trans* protocol is similar to that by Kaashoek above, except there is no single site. All messages are broadcast, so that all hosts can potentially receive every message, and the protocol uses both positive and negative acknowledgements. The basic premise for the operation of this protocol is that messages include acknowledgements, both positive and negative, as part of data messages. By including the control information in data messages, the number of messages transmitted is low, except in a lightly loaded system, where dummy messages are transmitted.

Each host maintains a list of received massages, a list of acknowledgements and a list of pending retransmissions, which are messages for which a negative acknowledgement was received. Because each message transmitted from the host contains a copy of the acknowledgement list, many hosts may attempt to retransmit a negatively acknowledged message. This is prevented by hosts observing broadcasts, and on the detection of a retransmission, removing the message identifier from the retransmission list. Most of the functionality of this protocol is in the adding and deleting of message identifiers from these three lists.

Because information about messages received by each host is transmitted then each host can determine firstly which messages have been received, and secondly the order in which they were received. By enumeration the *Observable Predicate of Delivery* can be used to determine when a message has been received by all of the hosts, so that it can be deleted from the received list. The protocol builds a *partial order* from the information contained in messages, which is the same for all hosts, as they will all have received the same information, although some of the hosts may not have a complete view of that order, due to the omission of, rather than mis-order of, messages.

The *Total protocol* works on the partial order achieved by the Trans protocol to form a total order across the system. The Total protocol would be unnecessary if the communication were totally reliable as the partial order would then be the total order. The protocol generates no additional traffic but delays the delivery of packets until a sufficient number of subsequent packets have been received to guarantee global order. Messages that do not follow the partial order become *candidate* messages, with at most one candidate message from each originator. Total order is extended by including a set of candidate messages in the total order, the decision being voted on, voting being carried out by the *Trans* protocol.

The group management required by this design is not described, as it is assumed that recovery from failure or network partition is automatic. A similar system to that used by Kaashoek for recovering buffer space is used by this design.

2.6 Desired Order

The causal broadcast protocol, *CBCAST*, used by the *ISIS* toolkit is the only example of its kind in the literature. Two variants on this design have been included in the latest version of *ISIS*, the later version being referred to as *fast* *CBCAST*. The earlier, and more general, *CBCAST* uses *clabels* to yield a partial order on *CBCAST*s using an *ISIS* system wide algorithm. *Clabels* are of no relevance if two *CBCAST*s have no destinations in common. It should be noted here that in *ISIS* each intended recipient application is addressed individually, the set of recipients being represented by a list of recipients, so that a sub-set of recipients within a group can be addressed. *Clabel*s are restricted in their use by the notion of *potential causality* [La78a], where a broadcast B is said to be causally related to broadcast B' only if either both broadcasts had the same originator and B' was transmitted after B or B was delivered to the originator of B' before B' was transmitted. The use of *clabel*s is application dependent. The causal broadcast itself is achieved by transmitting, in addition to the current message, any undelivered messages which preceded the current message to the recipients of the current message, ensuring that the causal relationship holds even for messages which were not originally destined for that recipient. A number of optimizations can be made to this design [Bi87a]. Fast *CBCAST* uses a distributed timestamping algorithm to ensure causality that requires a more restricted use of multicast addresses than for the protocol described above. A number of variations on the timestamping theme are described by Birman, Schiper and Stephenson [Bi90a].

Causality is used within the context of the *ISIS* programming [Bi89b] environment as the primary method for data transfer, having a lower transmission cost than the alternatives while being subject to the virtual synchrony of the environment. The protocol is reliable, this being assumed for all of the *ISIS* protocols, and to some extent fault tolerant. The protocol is reliable because it uses the reliable byte stream service offered by *TCP* for communication between *ISIS* sites.

While causality here is described as being a separate ordering paradigm, the essence of causality is that a *desired* order may be defined for messages. In both the group ordering and overlapped group ordering schemes, the actual order is arbitrary, it only being guaranteed that the order achieved is consistent. Therefore, it is not inconceivable that a desired order be combined with these other ordering paradigms.

2.7 Summary

The usefulness of multicast in distributed programming is reflected in the use of the technique by many distributed operating systems, in conjunction with the Remote Procedure Call paradigm for communication abstraction. The technique is applicable to any application which exhibits replication, either of data or of the application itself.

A number of protocol designs have been discussed in the context of a taxonomy of multicast. The most important feature of this is the order in which messages are processed by group members, with a variety of ordering paradigms described. The similarity in approach in many of these protocols indicates that there are only a finite number of basic methods which apply to these systems, many of them exploiting specific characteristics of networks to operate correctly. The scalability characteristics of these designs are explored later.

Chapter 3
Multicast Routing

One of the factors favouring multicast over multiple unicasts, is the use of broadcast based networks by such protocols to reduce the number of messages transmitted. While such networks are relatively common in the LAN world, they are by no means the only architecture on which multicast is possible. The use of multiple unicasts for multicasting is used by many protocol designs, even where the underlying network architecture supports broadcast directly. While these two methods form the majority of cases, the efficiency of multiple unicasting may be increased by delaying the copying of packets until required.

A common architecture for both WAN and closely coupled multi-processor networks is based on message forwarding, where messages destined for recipients not directly connected to the originator have to be forwarded by some intermediate node. The path followed by such a message is referred to as the *route*, the path being dictated by a *routing protocol.*

In a WAN environment, where nodes are geographically dispersed, the problems of calculating the best path for messages to follow from originator to recipients requires the use of a distributed routing scheme, each node of the network gathering information from immediate neighbours about the state of the network. In closely coupled networks of processors a routing graph is used by each node to indicate the path to take to each of the other processors, which can be calculated easily as the complete topology of the system is available and also the reliability of the components is generally higher, reducing the need for topology updates.

Multicast routing differs from unicast in that there is more than one routing path a message can take, with only one copy of the message received by the recipients. The message can be said to *explode* at each split in the multicast route. The explosion can take place at any point in the route, from the originator to the penultimate node. Message explosion at the originator is just multiple unicast.

3.1 Wide Area Routing

A number of multicast extensions to WAN protocols have been designed, mostly for the IP, as this protocol already possesses a broadcast capability. Figure 3.1 shows the architecture of a simple store and forward network, the dotted lines denoting the path taken by a multicast.

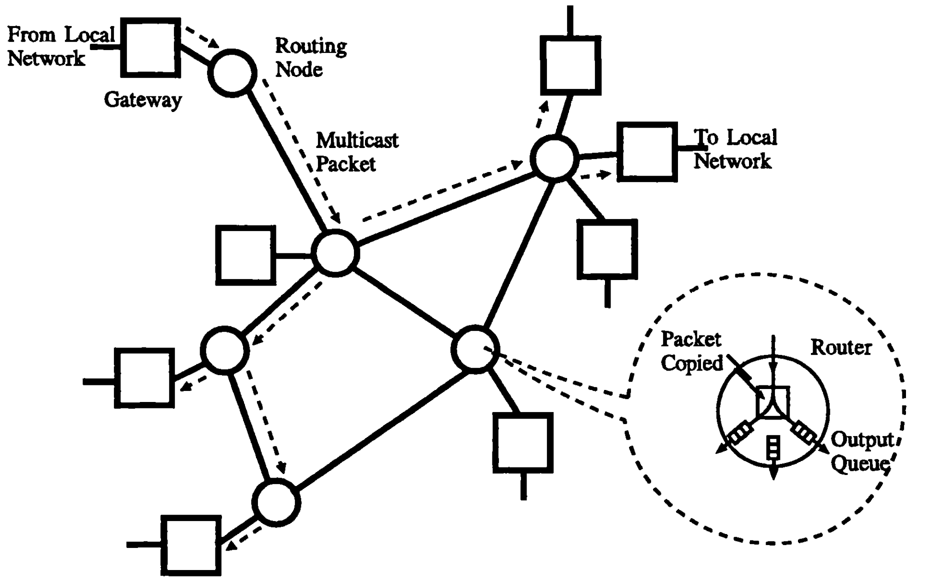


Figure 3.1 Multicast Routing in WAN Environment

A protocol modification to the IP by Aguilar [Ag84a] involved extending the IP options field [Po81a] to include a list of unicast addresses to which the message is destined. Up to 9 such addresses could be included in the IP header, 8 in the options field and the normal destination address, the datagram being termed a *multigram.* In the Internet, the destination address is used to route the datagram, so that by including a list of destination addresses *in the order in which they occur in the path from* *originator to final recipient*, as calculated using routing information, the multigram can be successively routed to all of the destinations. At each intermediate node the next node on the route is calculated using the current destination address field.

When a node that is considered local to the current destination address is reached, the multigram is replicated and the next destination is placed in the current destination field. This is also checked to see if it is also destined for this node. When an address is encountered which is not destined for local consumption, the multigram is forwarded as before, all local copies of the multigram being transmitted to the respective destinations. If more than 9 recipients are addressed then more than one multigram is generated by the originator.

Deering [De90a] describes more extensive modifications to two of the routing protocols used by the Internet, which enables true multicast routing of datagrams using multicast addresses. In addition, each multicasting host is required to operate a multicast extension to the *IP*, called the Internet Group Management Protocol (*IGMP*) [De89a]. It is interesting to note the change of emphasis of this protocol over time, the latest version being used primarily to allow the reception of multicast datagrams, previous versions of the protocol [De85a,De86a,De88a] possessing active group management for multicasting.

The two routing algorithms are the distance vector routing algorithm, also known as the Bellman-Ford or Ford-Fulkerson algorithm and the link state routing also known as the "New Arpanet" or "Shortest Path First" algorithm.

Distance Vector Routing

Routers using the distance vector algorithm maintain a routing table which contains an entry for every reachable destination in the network. Each entry has information about the distance in intermediate nodes (*hops*) to the destination and the next hop address. To prevent old information being used for routing each entry is given an *age*.

The tables are built and maintained by the routers by periodically sending a routing packet out on each of its incident links. For *LAN* links the routing packet is usually transmitted by a local broadcast or multicast in order to reach all neighbouring routers. The packet contains a list of (*destination, distance*) pairs (a distance vector) taken from the originating router's routing table. On receiving a routing packet, the recipient router may update its own table if the neighbour offers a better route to a destination, or if the old route is no longer available.

Four algorithms are described which are modified versions of Dalal and Metcalfe's Reverse Path Forwarding [Da78a] broadcast algorithm.

In the basic reverse path forwarding algorithm, a router forwards a broadcast packet originating at source *S* if and only if it arrives via the shortest path from the router back to *S* (the "reverse path"). The router forwards the packet out on all incident links except the one on which the packet arrived. In networks where the "length" of each path is the same in both directions, for example when using hop counts to measure path lengths, this algorithm results in a shortest path broadcast to all links.

To implement the basic reverse path forwarding algorithm a router must be able to identify the shortest path from the router back to any host. In internetworks that use distance vector routing for unicast traffic, that information is precisely what is stored in the routing tables of every router. In addition, most implementations of distance vector routing use hop counts as the distance measure as required by the algorithm.

To use this algorithm for multicast it is enough simply to specify a set of internetworks multicast addresses that can be used as packet destinations and perform reverse path forwarding on all packets destined for such addresses. Hosts choose which group they wish to belong to and discard any packets addressed to other groups.

The problem with the basic reverse path forwarding algorithm is that any single broadcast packet may traverse any link more than once, up to the number of routers that share the link.

To eliminate the duplicate broadcast packets generated by the *RPF* algorithm, it is necessary for each router to identify which of its links are "child" links in the shortest reverse path tree rooted at any given source *S*. Then, when a broadcast packet originating at *S* arrives by the shortest path back to *S*, the router can forward it out only on the "child" links.

Reverse Path Broadcasting (*RPB*) involves identifying a single "parent" router for each link, relative to each possible source *S*. The parent is the one with the minimum distance to *S*. In case of a tie the router with the lowest address is chosen. Over each of its links a particular router learns each neighbour's distance to every *S* - that is the information in the routing packets periodically transmitted. Therefore, each router can independently decide whether or not it is the parent of a particular link, relative to each *S*. The parent selection technique for eliminating duplicates requires one additional field in each routing table entry, which is a bit-map with one bit for each incident link.

Truncated Reverse Path Broadcasting (*TRPB*) prunes the tree created above so that the tree only reaches as far as the members of the destination group.

For a router to forgo forwarding a multicast packet over a leaf link that has no group members, the router must be able to (1) identify leaves and (2) detect group membership. Using the previous algorithm, a router can identify which of its links are child links, relative to a source *S*. Leaf links are simply those child links that no other router uses to reach *S*. If every router periodically sends a packet on each of its links, saying "This link is my next hop to these destinations", then the parent routers of those links can tell whether or not the links are leaves, for each possible destination.

To detect if there are group members on a particular leaf, the hosts periodically report their memberships. The routers then keep a list, for each incident link, of which groups are present on that link. In the routing tables another bit map, *leaves*, is used to identify which of the child links are leaf links.

In *TRPB* pruning the shortest path broadcast tree by sending membership reports to each multicast source is costly in bandwidth and router resources. As it is not expected that every source will be sending multicast packets to every group, a better method would be to prune only the multicast trees that are in use.

Reverse Path Multicasting provides *on demand pruning* of the shortest path multicast tree. When a source first sends a multicast packet to a group it is delivered along the shortest path broadcast tree to all links except non-member leaves by the *TRPB* algorithm. When the packet reaches a router for which all of the child links are leaves and none of them have members of the destination group, a *non-membership report* (*NMR*) for that *(source, group)* pair is generated and sent back to the router that is one hop towards the source. If the one hop back router receives *NMR*s from all of its child routers (that is, all routers on its child links that use those links to reach the source of the multicast), and if its child links also have no members, it in turn sends a non-membership report to its parent router. In this way the information about the locations of groups is propagated back to the source. Subsequent multicast transmissions from the source are then directed only to the links that have a member of the group on them.

A problem with this method is due to possible topology changes and the joining of a group by a new member. The latter case can be dealt with by the joining host sending a message to the local router informing it of its membership, this information being propagated when a source multicasts a message to the group. The former case would require the *RPM* router to cancel any outstanding *NMR*s to ensure a child link is included in any future multicast.

An implementation of a multicast routing protocol based on the Routing Information Protocol [He88a] is the Distance Vector Multicast Routing Protocol [Wa88a]. The protocol uses the Internet Group Management Protocol (*IGMP*) [De89a] to exchange routing datagrams. Communication with the *IP* routers to allow address discovery on multicast networks uses a modified Internet Control Message Protocol [De91a].

Link State Multicast Routing

Using this algorithm, every router monitors the state of each of its incident links (e.g. up/down, status, possibly traffic load etc). When the state of a link changes the routers attached to that link broadcast the new state to every other router in the internetwork. This broadcast is a special high priority flooding protocol that ensures every router learns of the state change quickly. Therefore, every router receives information about every other router and all links, from which each router can determine the complete topology of the internetwork. Having the complete topology each router can compute the shortest path tree rooted at itself. Using this tree the router determines the shortest path from itself to any destination for forwarding packets.

It is easy to extend this algorithm for multicast by having routers included as part of the state of a link, a list of multicast groups that have members on the link. When a new group appears, or an old one disappears, the routers attached to that link flood the new state as before. Again, having the complete topology means that the shortest path tree can be calculated, but this time the calculation is from any source to a group of receivers.

However, computing and keeping the complete state for every multicast group would consume significant resources, so borrowing the idea of on demand pruning of the *RPM* algorithm, each router uses a cache of multicast routing records. There is no need for an age to be placed on cache entries as the old records may be discarded when new entries need to be added to a full cache, possibly using a least recently used algorithm. Whenever the topology changes the entire cache is flushed and the information recomputed as needed. If a group is created or deleted on a link, all fields associated with that group and link are removed.

3.2 Local Area Networks

Routing can also apply to broadcast based *LAN*s [Fr85a,De90a] if the *LAN* is segmented by bridges. One of the problems of routing broadcast packets in a concatenated *LAN* is that of packet looping, that is a packet which circulates forever within the network. This can be prevented by either including a *hop count* in the packet header which is decremented every time it passes a bridge, the packet being discarded when the hop count reaches zero, or by ensuring that the network topology does not contain any loops either physically, or by using an algorithm to calculate a *single spanning tree* for the network, switching off forwarding in those bridges which cause a loop. A multicast packet can also be affected by looping, with the same arguments applying as for broadcast.

If the packets are multicast, then the traffic could be reduced if the bridges knew if any member of the destination group was present on an attached segment, inhibiting forwarding to that segment if no member were present. A single spanning tree for each multicast group could be constructed to this purpose. The problem is mainly one of informing the bridges of the existence of a particular group. An easy solution is for each group member to multicast to the bridge directly, possibly by using an *all bridges* multicast address as the destination and the group address as the source, with forwarding turned off for such packets, the reception of such a packet at a bridge interface informing the bridge of the presence of at least one multicast group member on that particular network segment. However, this can become costly if the number of group members on a segment is large. A modified scheme uses the multicast address as both the source and destination of a special packet. Using this the other group members on that segment receive the packet, inhibiting their own transmission of such a packet, while the bridge recognizes it as destined for it, by having the source and destination addresses the same. Both of these schemes would only have to operate when a group member is present on a segment, with the bridge ageing such information so that after a time period with no such packet, the entry for that segment is deleted, ensuring that the single spanning tree for that group does not include that segment.

3.3 Mixed LANs and WANs

In the real world *LAN*s and *WAN*s are connected. Using multicast in such an environment could cause problems due to possible unintended duplication of multicast addresses, and their subsequent propagation through a *WAN*. Using administered address could solve this problem, or using a global address service. Both of these approaches lack flexibility. One solution is to employ two multicasts, one for local consumption and one for wider use, both having to be transmitted if accessing a *WAN* group [Hu89a]. A problem here is that the *WAN* message is not limited in any way, so that it too will be propagated as far as possible. In a network that supports the notion of *hop counts*, such as the *IP*, the *scope* of such a message can be limited by the number of hops required [Wa80a]. This also removes the need for separate local and remote messages, as setting the hop count appropriately will automatically limit the distance the message can travel.

3.4 Closely Coupled Processor Networks

An illustration of the generality of multicast is its use in closely coupled multiprocessors, where the technique allows messages to be passed to a defined set of processors. Figure 3.2 shows a two-dimensional example of a closely-coupled processor architecture.

An example of a multicast protocol for use in such an environment is by Byrd et. al. [By87a]. The protocol uses a list of destinations for routing each packet, with the packet being exploded when required. Cut through routing is used to minimize the storage requirements at each node, with only enough storage to buffer one word provided. Control signals flow back to the originator indicating the state of the packet as the packet is transferred a word at a time. The deadlock avoidance scheme relies on the termination of a current transfer to break a potential deadlock.

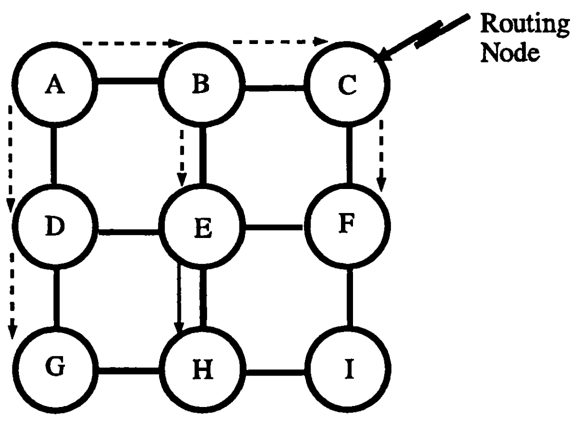


Figure 3.2 Routing in Closely Coupled Multiprocessor

A number of studies into the routing problems of closely coupled processor architectures use some kind of tree based routing for multicasting. Papers by Johnsson et. al.

[Jo89a], Dutt et. al. [Du90a], McKinley [Mc90a] and Kandler and Shin [Ka91a] are examples.

3.5 Summary

One scalability issue is the size of the environment, ranging from closely coupled multi-processors to national wide area networks. This reflects the general nature of multicast, which may be considered as a general communication model, encompassing both unicast and broadcast communication.

A number of methods for efficiently transmitting multicast messages in a *WAN* environment were described, based around the *IP*. The methods employed the information used by the common routing protocols for that architecture to reduce the overall packet count.

Routing within a concatenated *LAN* was discussed, where bridges employed a spanning tree to route packets. By allowing hosts to transmit to these bridges the multicast spanning tree could be pruned to reduce the number of packets forwarded to segments that did not have any active group members.

Chapter 4
What is Scalability?

Scalability has been described as a function of group size and influenced by the environment in which a multicast takes place. While group size may be viewed as a scalability measure, scalability is more about the limitations imposed on the group size supported by a protocol by problems such as implosion and the loss of packets in transit than on the actual size of the group.

Many multicast protocols have been discussed, most of which did not mention scalability as a significant factor in their design. For many protocols the issue was not considered because the protocol was designed for a particular scale and was therefore, implicitly, not to be used out of bounds. Others did not consider scalability an issue at all, not being relevant to the applications considered for multicast. While these points are valid, there is no way of knowing in advance the environment in which a protocol design may be used. A number of scalability issues are discussed with reference to those designs detailed previously. A model of implosion is described.

4.1 Issues of Scalability

While scalability is described as a function of group size, the effect that group size has depends on the issues described in chapter 2. Three of these are considered to be design issues, reliability, order and group management.

Group management impacts on the scalability of multicast because of the relationship between the number of group members and the amount of management information maintained. Scalability is therefore affected by the storage requirements of this information, which may be physically limited in some way, and by the need to maintain the management information is such a way to ensure that the protocol operates properly, which may be limited by the data transfer capabilities of the protocol that supports the group management.

Order impacts on scalability mainly because of the extra overhead imposed on the normal transfer of data to ensure the various ordering paradigms are maintained. Therefore, the main limitation is the actual data transfer process, which may be effectively reduced by this extra overhead and may therefore be substantially subsumed into considering the scalability of data transfer.

Reliability impacts on scalability because of the use of replies by reliability mechanisms and is the area investigated here. Because data transfer occurs more often than group management, the scalability of this is of relatively greater importance, and also because, apart from the storage aspect of management information, it is the maintenance of the management information that impacts on scalability, which may also be considered a data transfer issue as maintenance implies some form of data transfer between a holder of such information and the requestor. Reliability implies that replies are transmitted from recipients to the originator in response to the reception of a forward multicast. A number of points may be made about these replies. Firstly, it is likely that the time between the reception of a multicast message and the subsequent transmission of the reply will be similar for each of the recipients. Secondly, each recipient will have to reply even if only one reply is actually required, because each recipient in general will have no knowledge about the status of other recipients. This leads to the observation that if there are a large number of such recipients, and that the network connecting them to the originator exhibits a low dispersion, that is the difference in reception time between the "first" and "last" recipient is small, then there will be a proportionally large number of replies attempted in a relatively short time period. Of course, the network imposes its own mediation policy on these transmissions so that the replies are in effect transmitted sequentially, the minimum gap enforced by the network being used. If the processing rate of the host receiving these replies is less than the rate at which the replies are received from the network then there is a possibility of replies being missed. Buffering is normally used to absorb this rate mismatch. However, the number of these replies may cause this buffering to fill, again resulting in replies being missed. It is this *buffer overrun* that is considered to be the most important issue in the scalability of multicast.

Part of this buffer overrun problem is caused by the synchronizing effect of a multicast, which is effected by the dispersion experienced by packets passing through the network, typically low for LANs and MANs. Dispersion for multicast may be described as the difference in the times of arrival of a message at the group between the first message copy and the last. Multicast over WANs exhibits a greater degree of dispersion, reducing this problem, although DeSimone [De91b] shows that such networks possess correlation effects causing packets to cluster at routing hosts, a clear sign of possible congestion and buffer overrun.

The problem of implosion is analysed in more detail below. A related issue, affecting the transmission policy of a protocol, is also dependent on the group size and concerns the loss of packets at recipients due to packet corruption in transit or by congestion at the recipient or intermediate bridges. As the group size increases, it is more likely that group members will be in this situation, as a multicast also has to compete with other traffic. A simple model of this was developed [Jo91a], which assumed that each host possessed a probability of missing a packet, the accumulation of these probabilities decreasing the overall probability of a packet being received by all of the members of a group. Of course, flow control may be imposed on this traffic, however many of the flow control mechanisms employed require replies, with the potential for implosion. In addition, if many originators attempt communication with a group simultaneously then flow control has not yet been established, leading to the possibility of implosion, although the synchronization necessary for this to occur is less likely than that generated by a multicast.

An obvious scalability issue relates to the number of groups that may be supported by a host, the main limitation being the number of multicast addresses which may be filtered on simultaneously. One problem here relates to the need to filter packets efficiently, only allowing packets for which interest is indicated to be passed higher up the protocol stack. If the number of such filters is small then either the number of groups supported must be restricted, or all multicast must be passed by the filter increasing the number of packets passed up, and therefore potentially increasing buffer overrun. If the number of filters is large, then the time required to compare the packet address with the filter may also cause an increase in buffer overrun. Also, if a large number of packets are received at a host the processing load may result in unacceptable performance from the host, a congestion issue.

4.2 Implosion

Congestion is common to unicast communications as well as multicast, and occurs whenever the number and rate of packets arriving at a host exceed the capability of the recipient to buffer temporarily the packets. The congestion is normally relieved by temporarily storing messages until they can be processed, buffering working because congestion is normally a temporary problem, in that the input pauses allowing time for the output to reduce the buffer storage. However, a pause may not occur early enough to prevent packets being discarded. With the transfer rates of networks increasing rapidly, an increase that is not matched by similar increases in the speeds of other components, such as computer buses and processors, buffer overrun is expected to become even more of a problem, not just for multicast, but also for unicast. However, for positively acknowledged and transactional multicast protocols the problem is not controlling the flow from originator to recipients, but the control of replies, especially positive acknowledgements, which are not traditionally flow controlled.

The mechanism for buffer overrun in multicast is based on time. A transactional or positively acknowledged protocol design is compelled to return a message of some form to the originator in response to a multicast. Many protocols require that this reply be transmitted *as soon as possible*, after the reception of a multicast. Therefore, if each recipient receives the multicast within a small period of time, as will happen on a broadcast based *LAN*, then each recipient will attempt to reply, also within a short time period, given a certain amount of variability in the processing time of each recipient. It is here that dispersion is significant as it implies that if the dispersion is smaller than the time required to transmit a packet over the network then when the first reply is transmitted by a recipient, all of the other replies will be queued waiting for access. The network architecture imposes rules of its own on these replies to ensure that packets do not collide on the network itself, which implies that packets are transmitted as soon as possible by each recipient, given the network constraints of the minimum inter-packet gap. Therefore, the characteristics of the network and the number of recipients replying combine to potentially cause implosion which in turn may result in buffer overrun and the loss of replies.

Some argue that buffer overrun is not a problem, as a protocol can be designed to reduce the number of acknowledgements, others arguing that acknowledgements can be missed as only the first response is required. Although both of these can be justified in a limited context the problem of buffer overrun must be addressed in the design of a multicast protocol.

Danzig [Da89a] investigated the implosion effect at length, using a statistical representation of a typical network with attached hosts. The statistical model identified three buffer overrun points, figure 4.1, within a host.

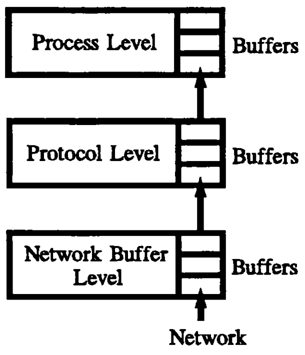


Figure 4.1 Danzig's Host Model

Each buffer overrun point has an associated *overhead* between the reception of a message and the subsequent removal of that message, either by passing it to the next level, or by being destroyed after processing completion.

Implosion does not just affect hosts, being also a problem for certain network architectures, specifically those which use some form of collision to resolve media conflict. These are generically titled Carrier Sense Multiple Access networks, being exemplified by the Ethernet [Me76a,Bo88a,Bo89a] network. Implosion affects these networks because if many hosts attempt to access the network within a short *collision period* their respective packets interfere with each other. The Ethernet protocol senses this collision, each host then stopping their transmission and executing an algorithm to calculate a retransmission interval. One of the more serious, from the implosion point of view, is that network's ability to discard packets which have had a number of attempts at transmission which resulted in a collision. The effect of multicast on an Ethernet was investigated, statistically, by Crowcroft and Paliwoda [Cr88a], the subsequent protocol design used a delay algorithm to increase the efficiency of the protocol.

It can be argued that buffer overrun can be dealt with by increasing the number of buffers available for the gathering of replies. However, adding extra buffering to hosts is only a temporary measure, as soon as the combination of network characteristics and number of recipients exceeds the new limit the problem will re-occur.

In order to investigate scalability, the parameters of implosion need to be mapped, so that these limits may be explored. Having described where implosion occurs, methods of overcoming implosion, that is using a protocol method to combat implosion, may be effectively employed. Given that the cause of implosion is a combination of the network environment and the number of replies exceeding the buffering capabilities of a host, three methods may be employed. The first would be to apply those flow control techniques described previously to the replies, as already discussed flow control is not traditionally applied to positive acknowledgements for the simple reason that often the acknowledgement is the flow control mechanism. The second is to reduce the number of replies to the host so that the buffering is able to cope. The third is to increase the time between replies to give the host more time to process already received messages.

4.3 A Review of Protocol Designs

The protocols described in Chapter 2 were, in general, not designed for scalability. Although many of them discuss their performance with respect to the number of participants, describing this as scalability, the number of participants is usually small, in the region of tens of hosts. While the designs work and work well in this region, extrapolating to larger groups should be treated cautiously. It should be noted that implosion will *always* occur whenever a multicast is replied to either with an acknowledgement or by a reply message, although that implosion may not necessarily result in buffer overrun. While many of the protocols previously discussed were not specifically transactional, that does not preclude their use in such a manner by a higher protocol.

Centralized ordering protocols [Ka89a,Ch84a,Na88a,Ar92a,Ga88b,Ga88a], where a single designated group member is responsible for ordering messages from the other group members, have two potential scalability problems. The first is due to the potential for implosion at the central site, the second the performance of the central site in normal usage. The central site is an implosion point because of the messages that are directed exclusively to that site by all of the other group members. In addition, some of the protocol designs discussed require some form of buffer synchronization to ensure reliability, each group member being required to return the message number of the last message received in sequence so that the central site can flush any older messages from its buffers. The scalability limit here is based on the number of messages that the central site can process with an acceptable delay, which in turn relates to the number of such group members supported. Those protocol designs that are broadcast are more susceptible to this as the central site has to support all group members, rather than in some cases where there is a separate central site for each group.

One of the problems of the central site concept is that of the failure of the central site, which may result in an inconsistent group state due to the loss of buffered, but not yet transmitted, packets being lost. This is a problem for those designs which acknowledge each message separately, which is the case if the group accepts messages from non-group members, as the subsequent multicast itself is not receivable by the originator, requiring an extra acknowledgement. Also, the failure of the central site requires that a new central site be created, most often employing some form of election algorithm to resolve any contention. As the central site also supports the group management information, then either this information must be maintained by every group member or be gathered when required, which is again a possible source of scalability problems.

Garcia-Molina's Propagation Graph, *PG*, produces a number of short trees, the structure of which being passed to the individual group members in order for them to forward messages based on the tree structure. Each membership change requires a new tree structure be calculated and distributed, a potential costly operation. One of the problems is that the "old" tree is used to propagate the "new" tree, which seems to assume that all parts of a tree are reachable at all times, something which may not in fact be true if the group change was prompted by a member failure. As the protocol was designed to be employed over an arbitrary network architecture there may be no recourse to a broadcast to inform any lost branches of the new graph, requiring the generator of the *PG* to directly inform the lost branch of the new graph.

Kaashoek's protocol [Ka89a] was reliable because the central site retained every packet transmitted to it, with packets being flushed from the finite buffer space when the central site was assured that every group member, in this case every host which was participating in the protocol, had received that packet. By implication therefore, some method was required to track the group membership, as the primary technique of flushing this buffer was to record the last received packet for each host, the last received sequence number being placed in each data packet by originating hosts. If the buffer became full, then a special flushing algorithm was used, which required each host to reply to a multicast. Two areas effectively limit the scalability of this design, firstly the buffering available, more hosts participating leading to potentially greater buffering requirements if the second limiting factor, the implosion of replies at the central site, is to be reduced.

Chang and Maxemchuk's [Ch84a] design employed a ring structure to ensure both that order was preserved and to protect against member failure. With increasing group size this ring becomes large, and because the design requires that each member be the central site twice before messages are delivered, the latency, that is the delay between transmission and delivery, increases proportionally.

Multi-phase and **positively acknowledged** protocols [Ve89a,Bi89b,Da89a] require that replies be received by each group member in order for the protocol to operate correctly, a classic example of the potential for implosion described above. Indeed, the requirement of multi-phase designs that this occurs a multiplicity of times exacerbates the situation. Because all the replies must be gathered, the originator must have complete knowledge of the group membership so that each reply can be "ticked off". The method for gathering this management information is often ignored in the design of such a protocol, being assumed to exist outside the protocol. Several methods have been proposed, such as providing a list of potential group members, which is subsequently pruned based on the replies gathered by a multicast or series of multicasts. The *ISIS* toolkit provides management information at each *ISIS* site, so that complete information is available, the information being maintained in a consistent and timely manner by the group management protocol, *GBCAST*.

The *AMp* [Ve89a], in addition to requiring that all replies be gathered at each phase, requires that all the replies are received in response to the same multicast message, ensuring that the multicast is atomic by guaranteeing that the multicast group receive the same message. This further restricts the potential scalability, as replies cannot be gathered over a number of transmissions.

A potential limiting factor may be seen in Danzig's protocol [Da89a] where a bit map of received acknowledgements is transmitted as part of a retransmission, the number of group members being limited by the size of this bit map. Of course, a bit map is the most efficient method of representing information and so can be extended without impacting too heavily on the data capacity of a packet, however a more sophisticated management policy must be employed to ensure that each group member "knows" which bit map field is allocated to itself, this information having to be passed to any originator on request.

Transactional protocols [Cr88a,Ch88a], where the reply contains data are also potentially limited by implosion. However, the number of replies that are actually required by the application making the multicast, expressed in terms of reliability, would tend to reduce the effect of buffer overrun if it occurs. If the transaction is set to be at least k-reliable, that is at least k replies are required, then as long as the buffer overrun occurs after k replies then there is no problem. Indeed Cheriton assumes in the context of the *V-Kernel* that only the first reply is needed, which assumes that the data contained in each reply is identical.

As has already been described, some failure modes may lead to replies which do not contain the same data, which if the above argument was employed may result in an inconsistent application state. In addition, many applications may require all replies in order to be assured that the latest, or most accurate data is used by the application. So, while transactional designs may be less effected by implosion, it is by no means certain that they will not be.

Simple protocols, such as those that use negative acknowledgements [Er87a,Me90a] or simple datagrams [Po80a] are less prone to reply synchronization, as replies are only generated by the detection of missed packets, or none at all. However, a number of shortcomings are inherent in such a system, the main one being the lack of reliability.

Although the use of negative acknowledgements reduces reply traffic, some synchronization will probably occur if messages are missed at bridges, so that a number of group members miss the same message, each then transmitting a *NACK*. One of the features of many negatively acknowledged designs is the use of extra traffic which is transmitted when there is no data to send, these extra messages being used for informing other group members of the last message received as well as the next sequence number to be used by that host. As a group becomes larger, the amount of overhead will also increase, possibly resulting in a substantial amount of unproductive activity.

The protocol by Melliar-Smith and others [Me90a] employs a distributed method of achieving order, each message containing information about messages received and the order in which they are currently arranged. It would be expected that as the group increases in size, the amount of this information may become a large proportion of the data in a message effectively limiting the number of group members supported.

Most of the protocol designs described in chapter 2 are designed around a broadcast capable *LAN* environment. One of the considerations when discussing large scale multicasting is the use of *WAN* to link *LAN*s together. A *WAN* can be characterized as having many paths to each recipient, of different lengths, which introduces dispersion into the system, as is artificially introduced using a delay algorithm. This is of benefit for multicast transactional protocols, reducing implosion at intermediate bridges and gateways, although as the speed of networks increase this dispersion is likely to reduce, as it is largely caused by storing message temporarily at bridges. However, the use of disparate paths, implying that messages are forwarded effectively in parallel, may be a cause of implosion. As implosion is caused mainly by a speed mismatch, then it is unlikely that messages passing from *WAN* to *LAN* implode, as a *LAN* is often very much faster than a *WAN*. In addition, buffer overrun at intermediate bridges may be delayed due to the dedicated nature of bridges, both in terms of the dedicated processing available for protocol processing and the large number of message buffers which can be expected.

4.4 A Model of Implosion

Implosion has been related to both a notional host and a real network. The notional host model by Danzig [Da89a] did not reflect the architecture of a real host in any detail. That model was based on three queues, figure 4.1, one at the network level, one at the protocol level and one at the process level. Each queue possessed a delay representing some form of processing. The model was driven by a network that was assumed to possess an infinite transfer rate, the only factor being the inter-arrival time of packets. The use of an infinite transfer rate was assumed to increase the implosion, and was therefore considered justified, with real systems exhibiting less implosion than predicted.

The Danzig host model was extended for this investigation and modified to relate more to real systems. Also, the scalability of multicast on an Ethernet [Me76a] was investigated, given that network's congestion control mechanism. The host model used here extended Danzig's model by including a bus level, as well as using a more realistic model for a network, Danzig using a network model which assumed an infinite transfer rate, so that only the inter-packet gap was considered significant. Figure 4.2 shows the modified host model used for simulation.

The model makes a number of assumption about the process of buffer overrun. Buffer overrun can have two causes. The first being the one investigated here, in which buffer overrun is considered as being caused by the inability of the host the consume packets. The second is considered to be caused by the inability of the application to consume packets, resulting in the cascading of buffer overrun from the top of the model to the bottom.

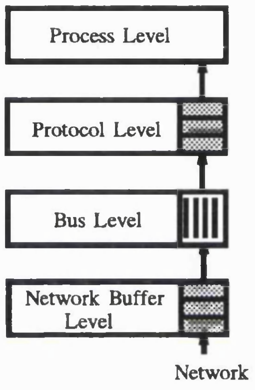


Figure 4.2 Extended Host Model

The difference is mainly of emphasis, the intention here to investigate how the architecture of a host affects buffer overrun. Thus, rather than look at buffer overrun directly, that is set a buffer size and observe when packets are discarded, each level has infinite buffering, and the measure taken is the rate at which packets are buffered. Therefore, no actual buffer overrun occurs, the intention being to measure the degree of buffer overrun which may then be related to the number of actual buffers used later.

The process level can be considered equivalent to the process abstraction of UNIX, or an application. Messages are transferred to the process or application using a *read* system call, commonly either copying the data from the protocol space to application space, or passing references to the buffer to the application. Although the time required for this transfer can be considered small in comparison with the other delays in the system, Cabrera et. al. [Ca88a] measured the characteristics of the interface between the operating system and process for UNIX 4.3BSD, finding that the interface between the kernel and the user process consumes around two thirds of the overall processing delay experienced by a packet. While memory to memory copying is fast compared to, for example, the time taken for a packet to be received off a typical network, the complications involved in ensuring that the process buffer was paged into main memory and the delay in actually making the process execute increased the overall time substantially.

The Danzig process level also included process buffering as a potential overflow point. While there is commonly buffering at the process, there are a number of reasons why such buffering may not be considered a particularly important overflow point. Firstly, extra buffers may be easily allocated, which given the large virtual address space offered by many 32-bit operating systems means that the number of such buffers can be assumed to reduce the problem. Secondly, as the application applies the reliability of the transaction, it can ensure that all possible replies can be buffered before making a multicast. Thirdly, if a process's buffering does fill, then the application may either ignore the reply, reuse a buffer or refuse the reply, resulting in the replies being stored in the protocol level until the application accepts. Because of the difficulty in quantifying the requirements for process buffering, it was assumed that all replies were receivable by the process.

The process level was assumed to be software interrupt driven, for example the use of *signal* or *select* in the UNIX operating system, and that there was a finite delay between the reception of a message signalling the process and the process acting on the signal. It was further assumed that to increase efficiency, a process reads all messages buffered by the protocol level at that time and that no other signals can be delivered while that signal is being handled. The delay was intended to simulate the effects observed by Cabrera described above.

The protocol level was considered to reside within the operating system and therefore possessed limited buffering, and a limited ability to allocate more space. The processing delay experienced by the message was considered to be relatively fixed, as most of the delay was assumed to be dictated by header processing. The effect here was to ignore the sizes of messages in the processing delay. Because a range of values were chosen for this delay, this assumption is reasonable. Size related delays are the copying and checksumming of packets, the latter carried out to ensure data integrity. The checksumming of packets is often the focus of much attention, a number of algorithms being described by Braden et. al. [Br89a], although on *LAN* this checksumming is often disabled, it being assumed that any packet errors were detected by the checksums calculated by lower level protocols. The copying of the packet to the process level was again assumed to be subsumed into the processing delay. While many operating systems require that the data in a packet be copied individually into the group member's process virtual address space, recent developments have allowed a buffer in process space to be mapped into the operating system kernel, the packet being transferred directly into the buffer from the bus, bypassing the kernels own internal buffering system and avoiding the overhead involved in byte copying.

The Danzig model described the network level as being directly attached to the protocol level. However, the use of a device bus to interface between device and main memory is a common architecture in real computers, and is therefore included here.

The bus level possesses no buffering, but imposes a transfer delay between the network level and the protocol level proportional to the message size. A typical bus is expected to transfer data from many devices, using pre-emption and time-slicing to enable bus sharing. A physical bus was considered initially, where the transfer was characterized by the bus width, in bits, and clock rate. This was developed so that the bus was characterized by the actual transfer rate, the transfer rate referring to sustainable rates rather than burst rates, sustainable transfer rates generally being lower than burst rates. Because packets are assumed to be larger than the burst transfer size, typically set to be a single bus cycle in length, this assumption is valid. Of course, only one transfer from a particular device can be in progress at any instance.

The network buffer level interfaces to the network itself and can be characterized as mainly buffering. It was assumed that the network buffer level was capable of receiving messages at the speed of the network, and also that the network buffer was able to set up a transfer to main memory in parallel with the reception of a message, so that there is no extra delay imposed on the message by the network buffer, over that inherent in the system. Although the network device is capable of a degree of concurrency, it was assumed that an entire packet must be received from the network before it can be subsequently transferred to the protocol level, which allows for the requirements of error checking at this level. This approach assumes that there is no extra delay involved in the processing of any packets, for example protocol processing by the network level. Because a range of bus transfer rates were employed, the effect of delay at the network level may be included as part of this "transfer delay" so that extra delay does not need to be modelled at this level.

The above defines a "real" host attached to a conceptual network. The conceptual network was used to drive the host model, the direction of flow of packets being from bottom to top only. The network was considered to have a transfer rate and an inter-packet gap. Each packet was considered to possess a length, which was divided between a header and data portions, the header portion being stripped by the network buffer, before transfer over the bus. As the header part is assumed to be stripped by the network level and also be relatively fixed in size across architectures, following the philosophy that the lower the protocol level the smaller the overhead, only the data length of a packet was variable. Although the header was fixed, it was not excluded from the model, it being expected that the effect of a small transfer delay, present on the network but not in a bus transfer, would effect implosion at the network level.

An implosion effect was also considered for an Ethernet, which is based on a distributed media access algorithm which uses Carrier Sense to detect the presence or absence of messages in transfer, and Collision Detection to detect when messages collide on the network. The relevance of implosion on this architecture is due to the ability of the network interface to discard a message if it has collided sixteen times. A collision occurs if two or more hosts transmit a message within a *collision period*, which occurs because of the propagation delay between a message being placed on the network and its detection by other hosts. A collision results in the colliding stations executing a binary backoff algorithm which forces the host to retransmit later, the delay increasing with each collision. A message is not transmitted if a transferring message is detected, the transmission being delayed until the current transmission ceases.

The models above were arrived at after considering a number of possible approaches to testing scalability.

One approach to designing for scale would be to design a protocol and, using the host model described above, investigate the scalability of the design directly, using a real network and real hosts. A number of factors preclude this approach, not only the lack of control over the environment, but also the need for a large number of hosts for testing. Given the experimental nature of any design this is not feasible on a working network, although a small amount of testing may be carried out to test the concept.

A more workable solution would be to implement the design under a network simulator, thus allowing absolute control of the environment and allowing access to all levels and parameters. However, a simulation is just that, a simulation, and cannot be considered a real substitute for actual testing. However, a simulator can show trends, and to a certain extent, the expected performance of the design under a wide variety of conditions, a number of host models being connected to a simulated network serving as a real network replacement.

While this approach has many merits, for the purpose of investigating implosion it was considered unnecessarily complex. A simplified protocol could be used, which just displayed the generic attributes of a technique. For example, a positively acknowledged design *must* acknowledge the last packet of a message, so the scalability of positive acknowledgements can be captured by just investigating this simple interaction. This approach was initially pursued, but, in turn found to offer no real benefit. After all, the aim is to investigate scalability, not any one protocol design, and implosion was identified as a major scalability parameter.

Finally, the host model and the network were separated, and investigated individually. As implosion most often occurs when acknowledgements, or replies are transmitted back to the originator, it is this reverse flow that was modelled. Because there was no longer any need for a protocol as such, the protocol level degenerated to a simple processing delay. The host model was forced using a separate network, as described by Danzig, although a more accurate portrayal of a network was used. Danzig assumed that the host was fed by a network that possessed no delay between the beginning and end of a message. In so doing the implosion was expected to be greater than in actuality, as there is a finite time required to transfer a message, giving each level extra time to process the previous messages.

The intention of this model is to investigate the parameters of implosion, which may then be applied to protocol design. It has already been seen how implosion is related to the protocol technique and the number and transfer rate of replies. These may now be related in turn to host architecture.

4.5 Summary

A number of issues were introduced as affecting the scalability of multicast protocols. Implosion of replies is considered a potential problem at the originator possibly leading to packet loss through buffer overrun. Implosion was described in greater detail, with reference to a host model.

The protocol designs described earlier were analysed for specific limitations in their scalability, the protocols being collected into groups based on the similarities in technique, which in turn exhibit similar scalability problems.

A modified host model was developed which used a more realistic network model than described previously, as well as a more physical representation of a typical host. This model was described in detail, and was used subsequently for simulation studies.

Chapter 5
Simulation

The host and Ethernet models which were developed to investigate the parameters of implosion were simulated using a propriety network simulator [So91a]. Simulation was employed for a number of reasons, most importantly the ability to control the required parameters. The simulator chosen used graphical output enabling the functionality of the simulation modules to be verified, aiding in the diagnostic and debugging phase. The simulator proper offered a number of basic building blocks, although the design had to be modified to allow its use for multicast.

5.1 Simulation

The reasons for choosing simulation over other methods for determining implosion for a range of network and host characteristics were described earlier. The most important was the flexibility offered by simulation, allowing network and host to be controlled through a few parameters. Graphical output allowed the effects of parameter changes to be seen quickly, which also aided in the debugging of the simulation modules.

As implosion was considered a potential scalability issue for multicast protocols, the causes of potential implosion were reduced to a few parameters, related to a typical host model. In addition, a popular network architecture, the Ethernet, was also seen as possessing a potential scalability problem due to that architectures use of packet discarding when the load was high, something which may occur with multicast replies, forming a temporary excess of traffic. The main aim in testing this aspect of the Ethernet architecture was to discover whether any limit on the number of multicasting hosts was evident. In addition, Crowcroft and Paliwoda's [Cr88a] work indicated that if a small delay was imposed on replies on an Ethernet the completion time was reduced over the case where there was no imposed delay, a feature also investigated.

The major part of the simulation was centred on the host model already described. The aim here was to use this model to assess firstly the conditions that lead to implosion, and secondly to use the values output from the simulation to find some measure which may be applied to implosion which indicates the scalability of a particular host system. The graphical output was useful here as the development of the final graphs over time lead to the measure of implosion described later.

The simulation model reduced the number of parameters by modelling the system with infinite buffer space at each buffering level. The result was that on implosion, there was no buffer overrun exhibited by the model, the measurement being the rate at which buffers were consumed by the system. Measuring the rate at which buffers were consumed gave more information about how the system behaved than setting an arbitrary number of buffers and finding how the system behaved. Because the simulation was driven by a deterministic algorithm, the rate of buffer consumption reached a steady state, which led directly to the implosion measurement described later.

A deterministic algorithm was chosen to drive the simulation for a number of reasons. Firstly, the simulation was simpler and enabled a direct relationship between the rate of buffer consumption and driving parameters to be established. Secondly, such an algorithm is more representative of more recent network architectures, such as token rings, where access to the network is more controlled than for Ethernet. Lastly, it was considered that a more variant algorithm would not be any more useful than that used, as it would be unlikely to reflect a real network's characteristics with real traffic. Recording and subsequently playing back such traffic was considered, but rejected as too complex for the purposes of this work.

5.2 Ethernet

Experiments on the simulated Ethernet were intended to locate the number of recipients which could be supported with a positively acknowledged protocol, and to find which of the many possible parameters affected this number. By charting the scalability of the Ethernet it was hoped that the applicability of many of the protocol designs described previously to large scale use can be determined for this network architecture. The Ethernet architecture was chosen for two reasons. The first is due to the design of the protocol itself, the protocol discarding packets if it is unable to transmit them, the second is the recognition of the large installed base of this type of network.

The Ethernet [Me76a,Bo88a,Bo89a] network architecture uses a backoff algorithm to mediate access to the shared media whenever a collision occurs. This algorithm discards any packet which has collided more than sixteen times. The discarding of packets under these conditions is a possible limit on the number of multicast recipients this network can support, as a multicast that results in replies or acknowledgements, will cause collisions due to the synchronizing effect of the multicast. By locating the maximum number of recipients which can be supported using a positively acknowledged protocol then the scalability of those designs, on an Ethernet, can be determined.

Four simulation modules were used for these experiments. The Ethernet module itself could have the length of the unbridged network segment altered, which affected the collision period of the network, that is the time between the start of a packet transmission and the time at which no collisions could occur as the transmission had reached the furthest part of the network, preventing transmission by any other host, the so called carrier sense part of the architecture. An associated module was the Ethernet Host module, which simulated that part of the Ethernet network which was considered host specific, such as the number of times a host had transmitted. There were no external parameters to this module. The operation of these two modules were modelled on the Ethernet standard.

The Host was simulated by a single module which generated network traffic, either externally based on time, or in response to a multicast. The main parameters of interest for this module were the packet length's of both the multicast and the reply, and the amount of delay imposed on replies, which used a uniform distribution to calculate delay.

The final module was a pure traffic generator, which was used to generate extra traffic over and above that generated by the modules described above. Figure 5.1 shows the module structure used for these experiments.

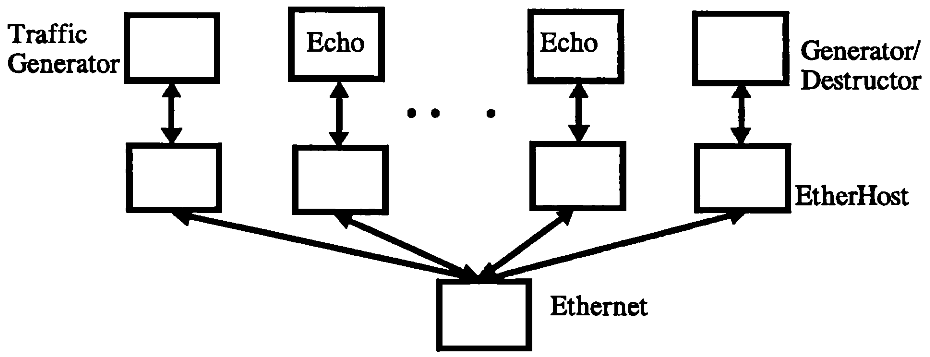


Figure 5.1 Experimental Configuration of Ethernet

The measures chosen for these experiments were the completion time, that is between the initial multicast and the gathering of the last reply, and the number of excessive collisions which occurred, resulting in the packet being dropped. The number of actual collisions that occurred was also considered and measured. While this could be used for investigating the reduction in collisions that occurred under the conditions described below, it is only excessive collisions which result in packets being discarded.

Because the simulation time was proportional to the number of recipients attached to the Ethernet, the number of data points generated for the larger group sizes was relatively small. The results were plotted by taking the average of the generated results, with a maximum and minimum to indicate bounds. The experimental procedure used one generator generating a multicast every second, this value being chosen after a number of experimental tests indicated that this time was larger than the largest completion time for the maximum group size considered. Recipients replied to this originator using a single packet reply, the number of such recipients being varied between ten and five hundred.

An initial experiment was carried out to set a baseline for the other experiments in order to compare and contrast the results obtained. The baseline was chosen such that each recipient replied *as soon as possible*, that is with no extra delay imposed on the reply. The message length was 1024 bytes, both multicast and reply, and an Ethernet length of 2500 metres. It should be noted that all length values *exclude* the size of the ethernet header.

Figure 5.2a shows the completion time experienced as a function of the number of recipients. The curve exhibits a knee at around 200 recipients. For recipient numbers of less than this, the curve is nearly linear. Figure 5.2b shows the number of excessive collisions, that is the number of transmission failures due to an excessive number of packet collisions, resulting in the failure of the multicast, with respect to the number of recipients. The reason for the knee above is now apparent. The number of recipients for which replies were received is less than expected due to discarded packets.

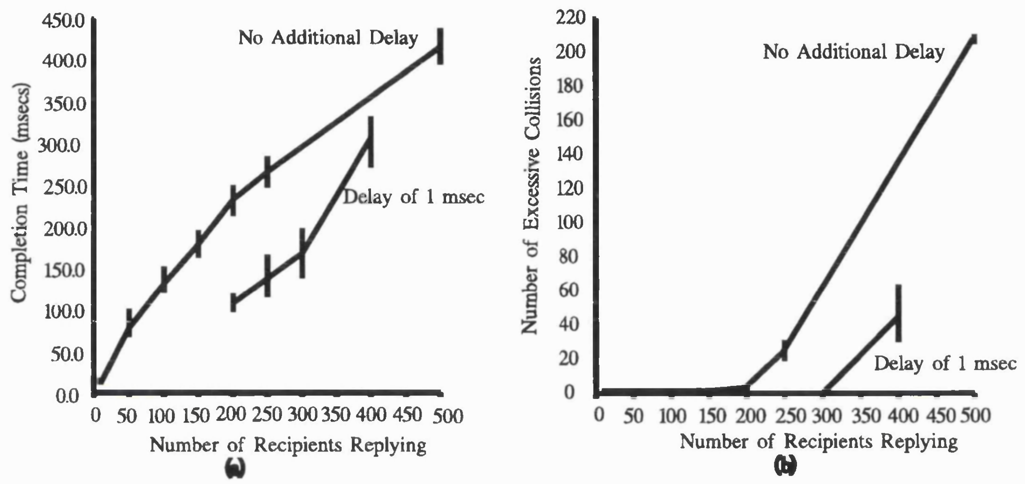


Figure 5.2 Testing of Ethernet of Length 2500 metres

Figure 5.3 shows the completion time when the number of packets lost due to excessive collisions is taken into account.

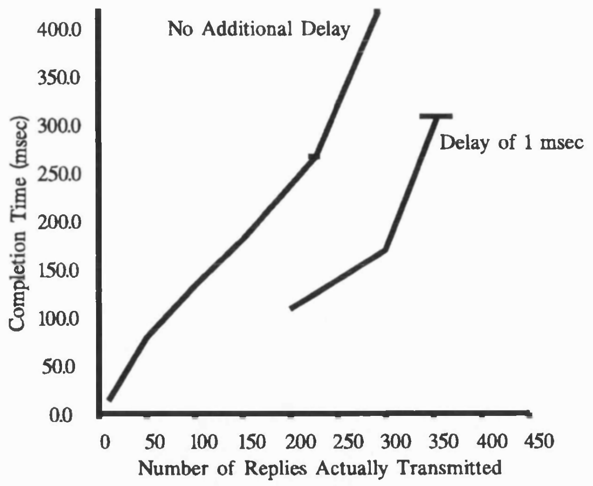
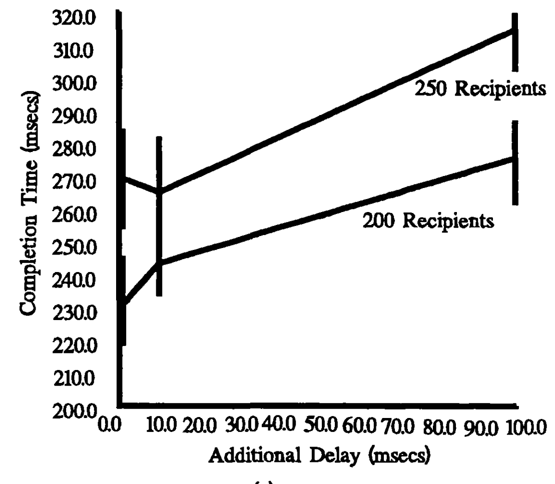


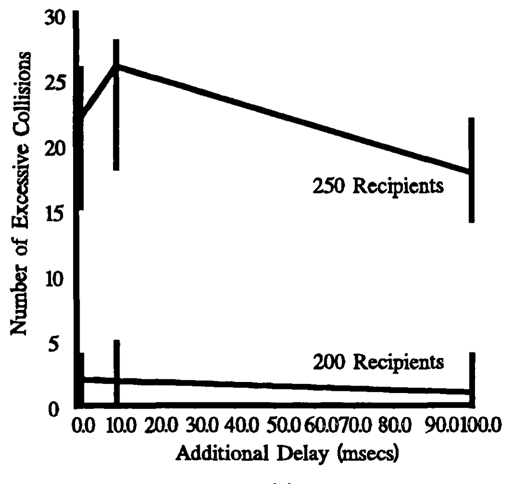
Figure 5.3 Normalized Completion Time vs. Number of Recipients

The main conclusion to be drawn here is that a multicast to between 150 and 200 recipients will have a high probability of failure, with multicasts to larger groups always failing. Of course this only applies to those protocols which require a reply from each recipient. The completion time for multicasts above this range is effectively infinite, as no completion occurs.

One method for overcoming implosion, and here which can be likened to a more heterogeneous environment, where hosts have differing reply times, involves using some kind of delay at each recipient to reduce contention for the media. Three values for this delay were chosen, 1 millisecond, 10 milliseconds and 100 milliseconds. As a baseline the corresponding value from the above graphs was also used. The first value can also be considered to reflect a heterogeneous environment populated by hosts which have a wide variety of delays as part of protocol processing. The delay values were used to form the mean and variance of a normal distribution. The aim here was to investigate if such means may change the effective limit described above, and therefore, to reduce the number of experiments carried out, the number of recipients were limited to 200 and 250 respectively. Figure 5.4a shows how these delays effected the completion times experienced in this range, with figure 5.4b showing how the number of excessive collisions were effected by the delays.



(a)

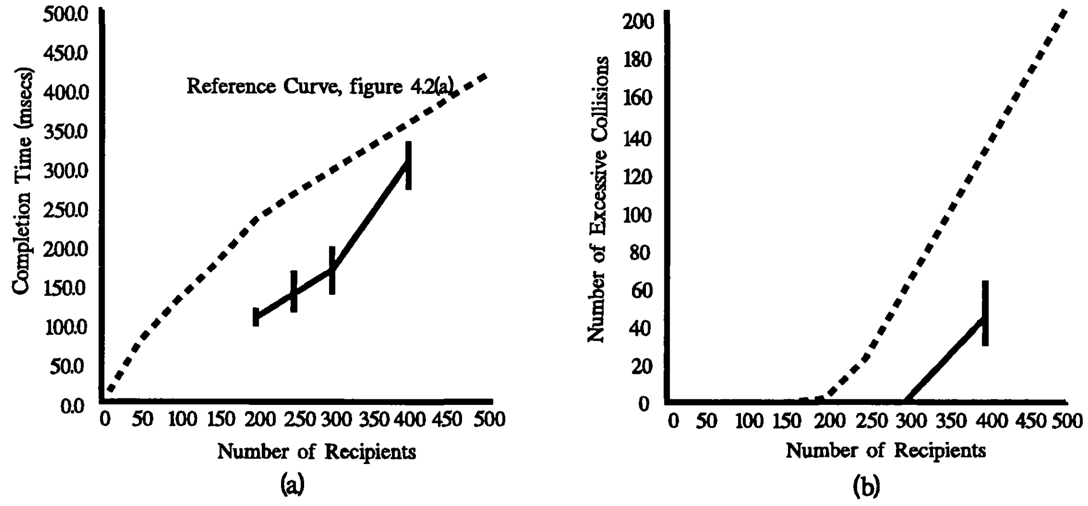


(b)

Figures 5.4 Effect of Random Delay on Completion Time and Excessive Collisions

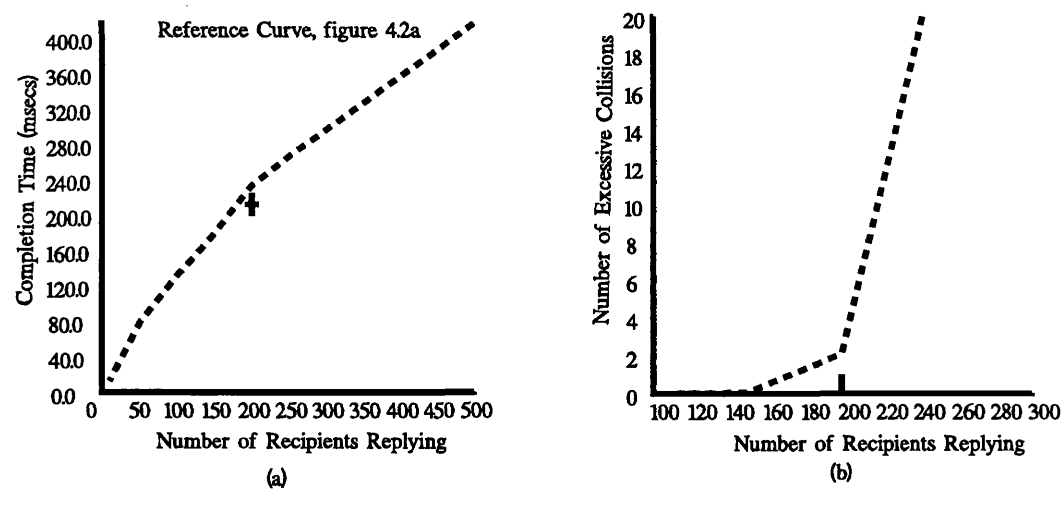
Comparing these values with the control curves, figures 5.2a and 5.2b respectively, it should be noted that the completion times increase in line with expectations, following the gradient of the curve. The most interesting feature is the reduction in excessive collisions as the delay increases. This indicates that a backoff strategy may be employed to reduce implosion, but at the expense of a greater completion time. It should be noted, but not shown, that the average numbers of collisions that occurred increased with increasing delay, resulting in a reduction in excessive collisions. This is counter intuitive, and seems to indicate that while more collisions occur, fewer recipients exceed the collisions limit.

Using the control curve as a reference, the effect of reply size on the implosion limit described above was investigated. The minimum size of packet which may be transmitted by an Ethernet is 46 bytes. This value formed the reply size used for these experiments. The multicast packet size remained 1024 bytes long. Figure 5.5a shows the completion time against number of recipients observed, figure 5.5b shows how the number of excessive collisions varied with group size.



Figures 5.5 Effect of Packet Size on Completion Time and Excessive Collisions

The results show that reply size has a positive effect on implosion, virtually doubling the observed limit.

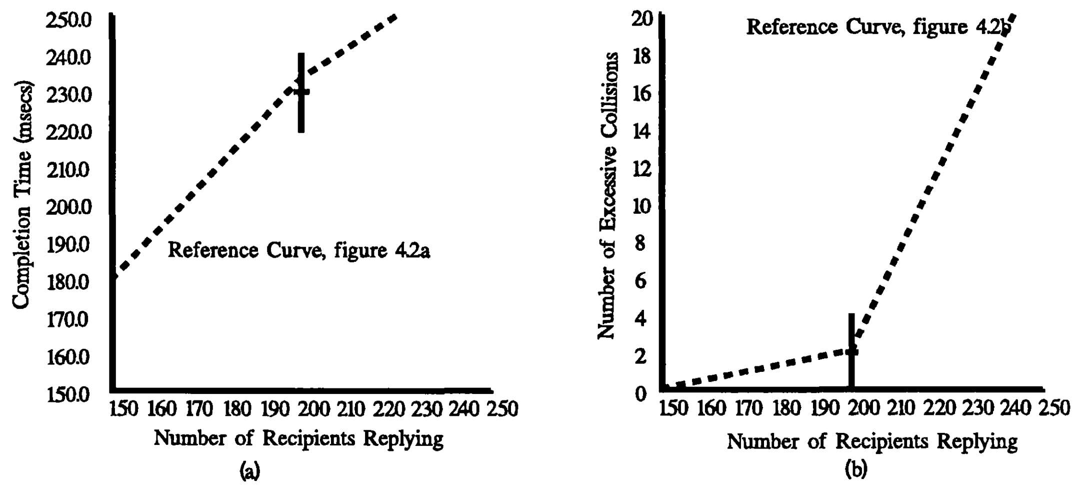


Figures 5.6 Effect of Length on Completion Time and Excessive Collisions

The effect of Ethernet length was investigated by setting the length parameter to 500 meters, and setting all other parameters to match the controls described above. Figure 5.6a shows the single result obtained with respect to the control curve, figure 5.6b showing how the number of excessive collisions is effected by Ethernet length with respect to the number of recipients.

The single result indicates that the length of the Ethernet has a positive effect on implosion. The length of the Ethernet cable affects the collision period, which seems to be reflected in the reduced number of excessive collisions, and also the reduced number of actual collisions.

Another parameter which potentially affected the limit described above was the presence of external traffic to the multicasting participants. No attempt was made to quantify the traffic level, as the intention was only to investigate the effect of traffic. Figure 5.7a shows how completion time was effected by traffic, figure 5.7b showing the effect on the number of excessive collisions with respect to the number of recipients due to external traffic.



Figures 5.7 Effect of Traffic on Tompletion Time and Excessive Collisions

These results indicate that completion time is reduced by the presence of traffic, although the number of excessive collisions was not.

The results above show that an Ethernet type of network exhibits a distinct failure pattern for multicast. Other network architectures are unlikely to suffer from this type of failure, as it requires a network architecture that drops messages which have been attempted a fixed number of times. The other area where messages are dropped is by buffer overrun in hosts, this problem being addressed in the next section.

5.3 Host

The host model described in Chapter 3 was used in order to investigate the effects of differing processing and network rate on a notional network host. In order to investigate implosion, rather than buffer overrun, it was decided to allow an effectively infinite number of buffers at the relevant levels in the host model, rather than a finite number of buffers. The former approach enables the rate of buffer increase to be determined, rather than the number of packets that caused a buffer overflow at a given rate. Each buffer was considered to be able to buffer an entire packet, so that it may be assumed for some architectures that smaller packets have more available buffering. A number of simulation modules were developed based on the structure of that host model. Figure 5.8 shows the experimental configuration.

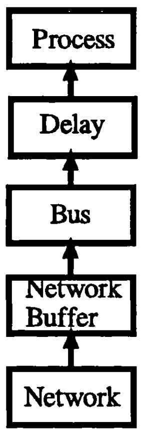


Figure 5.8 Experimental Host Configuration

The host was driven using the network module, which generated packets of variable length, that length being divided between a header and data portions. The header portion represented information used by the network itself, which was stripped from the packet in the network buffer module. The inter-packet gap and the transfer rate were used in conjunction with packet length to control the *effective transfer rate* of the network.

The network buffer module was used to buffer any packets which could not be transferred immediately over the bus module. Packets were considered to occupy a buffer between the time at which the start of the packet arrived, to the time at which the end of the packet was transferred to the bus module, and that only packets which had been fully received were eligible for transfer to the bus module. The network buffer module output buffer usage against time.

The bus module was used to impose a transfer delay related to the *sustained transfer rate*, measured in megabytes per second (*MBps*) and the number of data bytes transferred. The sustained transfer rate was used rather than a combination of the bus width and clock rate in order to normalize the parameters. By using a single transfer rate the overhead involved in setting up a transfer and the possible presence of other traffic were automatically allowed for. The values chosen for the transfer rates ranged from 10 *MBps* to 100 *MBps*, these values being based on some current and possible real values. The fast Small Computer Serial Interface (*SCSI*) standard is rated at around 10 *MBps*, with the *SBus* rate quoted at around 42 *MBps* in a *SPARC*station 2 [Bo91a].

The protocol module was used to impose a processing delay on each packet fully received by the module. The processing delay was considered fixed, reflecting packet header processing, rather than being related to the packet length. The delay values used were calculated with reference to developed from the work by Clark, Jacobsen, Romkey and Salwen [Cl89a] with respect to the *TCP*, where the emphasis was on finding and counting instruction path used for that protocol's normal operation, that is data transfer and the reception and processing of *PACK*s. Values of 100 VAX instructions for transmission and 120 for reception were presented. These figures were purely for protocol operations and did not reflect any of the buffer handling carried out on behalf of the protocol by the supporting operations system, nor the time needed to search for protocol control structures. For the purposes of this thesis a generic protocol was considered to require between 100 and 500 instructions for normal operations, an instruction being considered non-architecture specific for simplicity. In addition, Cabrera, Hunter, Karels and Mosher [Ca88a] investigated the characteristics of the UNIX Inter-Process communication method, which is the main method of accessing protocols. In order to generalize the values used, it was assumed that the above values included this processing overhead.

To demonstrate the effects of protocol processing time on multicast, the time taken to process any packet needs to be developed from the number of instructions. Such values are complicated by the indeterminate times required for fetching data and instructions from main memory, or from high speed memory caches if available, and also difficulty in calculating the number of clock cycles per instruction, typically ranging from about four to less than one for so called "super scalar" processors. Therefore, rather than attempt to calculate real values, notional figures are assumed, based on approximate values. A range from 5 million instructions per second (*MIPS*), reflecting a fast Personal Computer, to 200 *MIPS* was used, covering low end workstations to possible next generation *RISC* technology, which has already demonstrated the potential power of this architecture. Table 5.1 shows the processing delays experienced using a number of processing speeds.

MIPS	**per 100 instructions**	**per 500 instructions**
5	2.00×10^{-5}	1.0×10^{-4}
10	1.00×10^{-5}	5.00×10^{-5}
20	5.00×10^{-6}	2.50×10^{-5}
50	2.00×10^{-6}	1.00×10^{-5}
100	1.00×10^{-6}	5.00×10^{-6}
150	6.67×10^{-7}	3.33×10^{-6}
200	5.00×10^{-7}	2.50×10^{-6}

Table 5.1 Processing Speeds and Delays

From this table, a range of values were chosen which corresponded to those given in the table. Using time directly in this way reduced the number of variables as well as enabling more general conclusions to be drawn from the results.

Although the delays imposed by checksum processing and data copying, activities related to the number of data bytes in a message, were ignored here, in choosing a range of values the effects of such length related delays can be considered to be encapsulated within the given range of values, at least for the faster processing speeds.

The final module used here was the process module. The use of this module was intended to show the effect of a delay in reading buffered packets from protocol space, subject to an interrupt time. This interrupt time can be considered the time required for a software interrupt to be acted upon by the central processor, resulting in a process being either made runnable, so that a read can be made, or at least interrupt the process to execute the interrupt handler. This activity reflects a process that either waits for an interrupt to be raised, or carries out processing between interrupts and then reads packets when available. The other model which was considered was one where the process has already executed a read, and is waiting for packets to arrive, it being assumed that the read was intended to read the replies to a multicast and that the buffering for these replies *has already been allocated*, so that the protocol module only has to copy the data into these buffers, the process returning when the multicast completes. This model requires no interrupt time and is therefore better modelled by passing received packets directly to it after processing.

The host model can be considered in two sections. The first section concerns the network, network buffer and bus modules, the second the bus, protocol and process modules. The host model was divided in this way firstly to reduce the number of experiments required, by observing that whenever the network buffer overflowed, no more packets may be passed over the bus so that there will only be one value relevant in this situation, that being the bus transfer rate. In addition, separating these allows each potential overflow point to be investigated separately, to some extent. The main parameter used to measure this overflow is the rate of increase of buffer usage which is measured in *buffers per second.*

Network, Network Buffer and Bus

The Network module was driven using data transfer rates of 10, 100 and 1000 *Mbps* which covers many of the current network architectures. The packet header length was chosen to be 18 bytes. The data portion was varied between 8192 bytes and 64 bytes. The transfer rate of the network ranged from 10 Megabits per second (*Mbps*), characteristic of current network technology such as the Ethernet, to 1 Gigabit bit per second, 1000 *Mbps*, representing possible future research efforts in fibre optic communications. An intermediate rate of 100 *Mbps* was also considered, representative of current high speed networks, such as the Fibre Distributed Data Interface [Ro86a,Ro90a]. Inter-packet gaps can have a potentially large range, depending on the rate at which packets are placed onto a network by originators. However, for implosion values ranging from 1 millisecond (msec) to 0.1 microseconds (μsec) were chosen, intermediate values being 100μsecs, 10μsecs and 1μsec. These values were chosen in order to bracket the expected implosion points of the system, mainly the difference between the network rate and the bus transfer rate. For comparison the Ethernet exhibits a minimum inter-packet gap of 16μsecs, made up of a 9.6μsecs recovery time and a 5.4μsecs *preamble* which aids collision detection. As the Ethernet is an example of what may be termed old technology it is to be expected that the minimum gaps currently used are smaller, a minimum gap being assumed of 0.1μsecs. A uniform inter-packet gap models a more controlled environment than the Ethernet above, being more applicable to a token ring type of architecture. However, by choosing a range of inter-packet gaps, it is hoped that variation in the gaps between the replies may be extrapolated, making the experiments more representative, although it has already been noted that the synchronizing effect of a multicast may lead to replies being transmitted with the minimum gap allowed by a network architecture, especially if there are a large number of replies.

The experiments indicated that a network rate of 10 *Mbps* did not cause buffer overflow through the entire range of inter-message gaps, a buffer size of 2 being sufficient for this scenario. This was expected, as the slowest bus has a transfer rate greatly in excess of the network, even at maximum rate, with no inter-packet gap. A buffer size of two is a recurring theme throughout these results, being the maximum number of buffers used for a steady state condition.

A network rate of 100 *Mbps* exhibited overflow with a bus transfer rate of 10 *MBps*, as would be expected by normalizing the bus transfer rate to 80 *Mbps*. Figure 5.9 shows the relationship between the rate of buffer increase and the inter-message gap for that bus transfer rate.

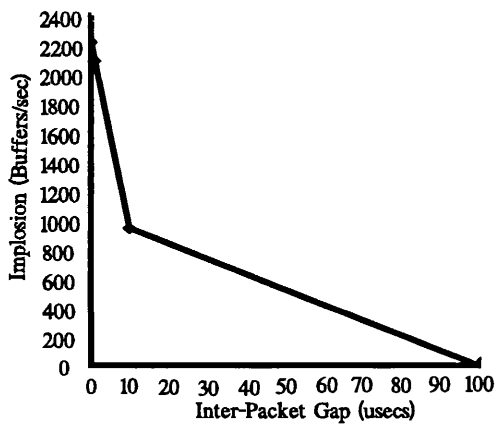


Figure 5.9 Rate of Buffer Increase for Bus Transfer Rate of 10 *MBps*

All other bus transfer rates exhibited no buffer overflow as above. It should be noted that the buffer increase was stepped, that is the buffer usage remained level with packets being input before the buffer usage increased. It is this stepping that suggested the possibility of interpreting the data differently, as will be described later.

Figure 5.10 shows buffer overflow for a network rate of 1000 *Mbps*, that is 1 *Gbps*, with respect to the inter-message gap for a range of bus transfer rates. Each of the bus transfer rates overflowed at some point. Many of these were also stepped, the size of each step being reflected in the error observed.

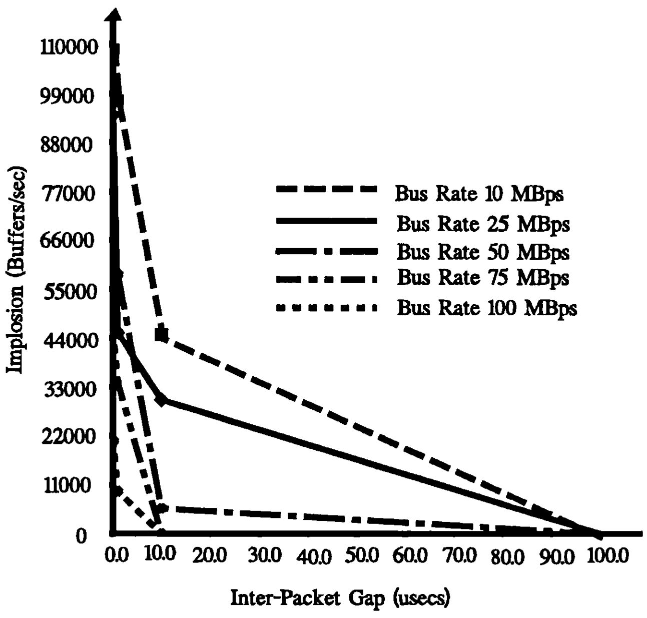


Figure 5.10 Rate of Buffer Increase for a Number of Bus Transfer Rates

Figure 5.11 shows the effect of varying the length of data with respect to the header size for a network transfer rate of 1000 *Mbps*, an inter-message gap of 10μsecs and a bus rate of 50 and 100 *MBps*.

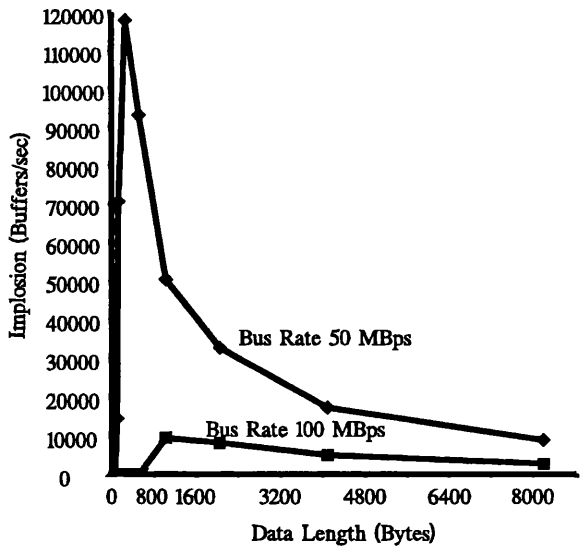


Figure 5.11 Effect of varying Data Length on Implosion

This result is interesting for its implications when the reply is an acknowledgement. Acknowledgements are generally short packets, which these results indicate are less affected by implosion than larger ones. Indeed, overflow does not occur for a packet size of 64 bytes, a typical acknowledgement size.

These results indicate the type of behaviour that may be expected of the interaction between bus and network. Because a typical network device has limited buffering, overrun here may be more severe than for the protocol buffering.

Bus, Protocol and Process

For the testing of these upper modules, the effect of network rate and gap was investigated for a number of processing delays. In the limit, where the network buffering is overflowing, the bus is unable to transfer any more packets than it is already doing and therefore any results in this region will be identical with respect to the bus transfer rate. In addition, where there is no network buffer overrun, then it is expected that the results for protocol buffering will be similar, as it is only the extra delay imposed by bus transfer which will effect the rate of buffer overrun.

Two sets of experiments were carried out, one set which assumed that messages were destroyed by the protocol module, the other where messages were handed to the process module. The latter was dependent on the time needed by a process to recognize that messages were queued awaiting reading by the process. The former represented either the normal destruction of an acknowledgement by the protocol or, the passing of a message to the process address space directly because the process was already waiting to read the message.

The protocol processing delay was set to 0.1 msecs, 0.01 msecs, 0.001 msecs and 0.0001 msecs, based on the values shown in table 5.1. The experimental conditions for the network were the same as above, that is a data length of 1024 bytes was used with the inter-packet gap ranging from 1 msec to 0.1μsec.

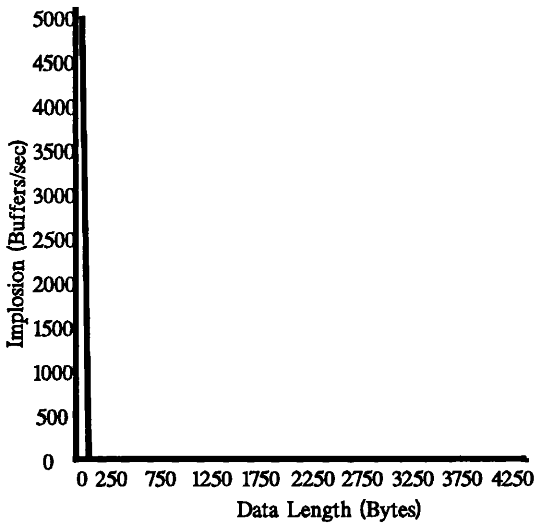


Figure 5.12 Effect of Packet Size on Implosion for 10 *Mbps* Network Transfer Rate

Testing using a network rate of 10 *Mbps* indicated that there was no overflow due to protocol processing delay for this packet size, indicating that current host architecture is well able to handle multicast replies on today's networks. The effect of the packet size was then investigated, the data portion of the packet being varied between 64 and 4096 bytes. The inter-packet gap was set to 1 μsec, figure 5.12 showing that implosion occurs when the packet size reduces and the protocol processing delay is 0. 1 msecs. Lower protocol processing delays exhibited no implosion.

A network rate of 100 *Mbps* was then tested. The results indicate that for a protocol processing delay of 0. 01 msec and below there was no implosion. However, a protocol processing delay of 0. 1 msecs did exhibit overflow, figure 5.13.

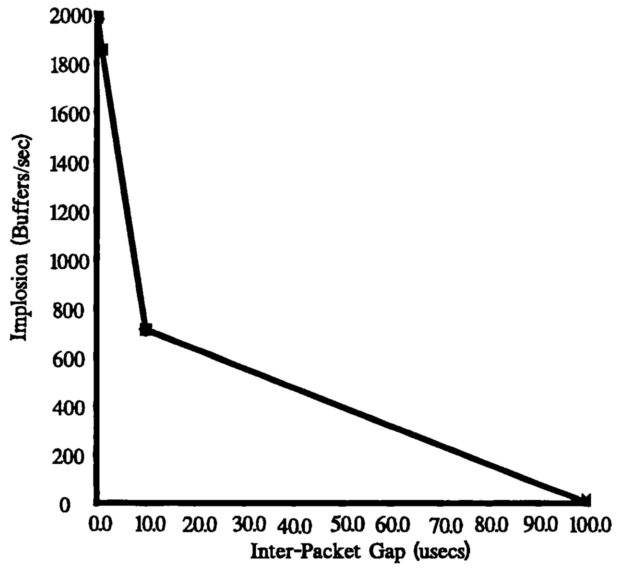


Figure 5.13 Implosion at Protocol Level

The similarities between these curves, for bus transfer rates of 25 *MBps* and above is due to the buffer overrun being protocol processing bound, the bus being well able to transfer packets, as evidenced by the lack of buffer overrun in the network buffer in the equivalent testing above. Using this observation it was felt that a reduction in the number of experiments could be achieved by concentrating on one bus transfer rate for all subsequent experiments. An interesting effect observed was for a bus transfer rate of 10 *MBps*, which corresponds with implosion at the network buffer level, figure 5.9. Here the rate of buffer increase was uniform for *all* of the protocol processing delays used. Indeed, this behaviour was noted whenever the bus was saturated, the actual rate of buffer increase being related to the bus transfer rate. Initially, the model, and the implementation of the model, was thought to be incorrect. However, careful analysis of the implementation, and the fact that the model operates as expected for other values seems to indicate some other mechanism for this.

A similar set of experiments were carried out for a network rate of 1 *Gbps*. Because every bus transfer rate caused bus saturation at some point, and the same behaviour described above occurred, no useful results were obtained.

One set of results which were gathered relate to a gap of 1μsec. Figure 5.14 shows how the rate of buffer increase increased with bus transfer rate in the bus saturation region. These results exhibit the constant overrun value described above, increasing with increasing bus transfer rate.

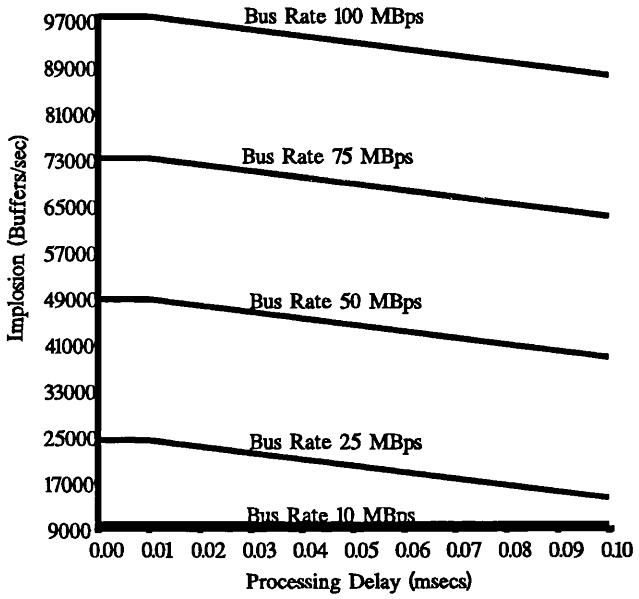


Figure 5.14 Implosion for 1 *Gbps* Network Rate

The length of the data in the packet was now varied, from 4096 bytes to 64 bytes.

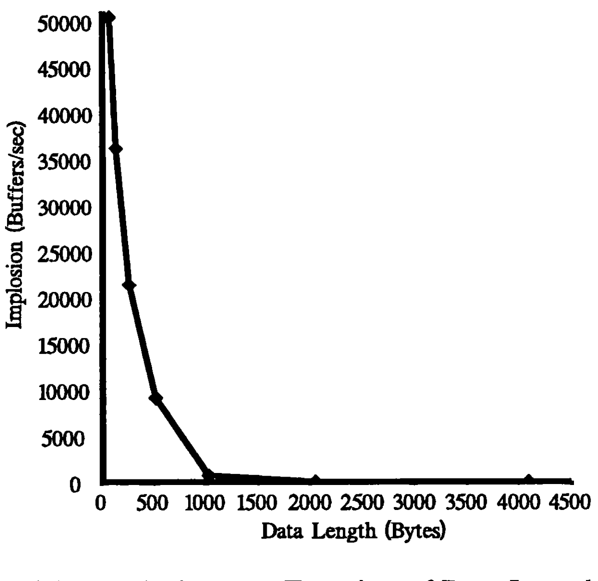


Figure 5.15 Implosion as a Function of Data Length

Figure 5.15 shows how the implosion varied when the network transfer rate was set to 100 *Mbps*, the inter-packet gap to 10μsecs and the bus transfer rate to 50 *MBps*.

No implosion occurred for protocol processing delays greater then 0.1 msecs. The curve indicates that implosion increases with decreasing packet size, which was expected as the protocol processing delay was assumed to be independent of packet length. The inter-packet gap was then set to 1μsecs for the same range of data lengths. Figure 5.16 shows two curves, one for a protocol processing delay of 0.1 msecs, the other of 0.01 msecs.

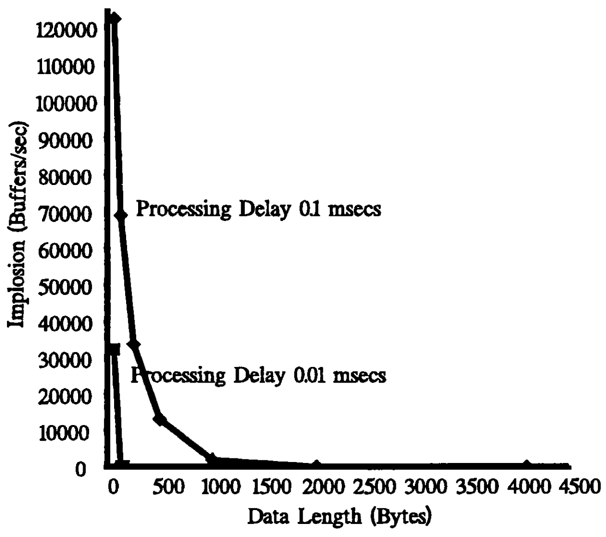


Figure 5.16 Implosion as a Function of Data Length for Gap of 1μsec

Protocol processing delays greater than 0.01 msecs exhibiting no implosion. The trend is for increasing implosion with decreasing packet length and inter-packet gap, the effect of which is to increase the number of packets arriving per second.

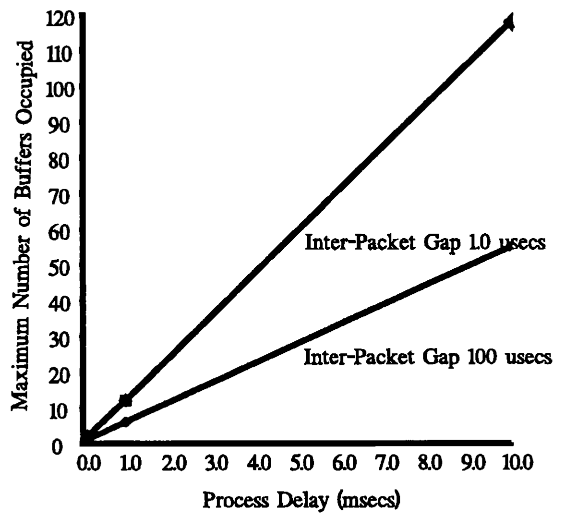


Figure 5.17 Effect of Processing Delay on Buffer Size

All of the above have assumed that packets are destroyed by the protocol. The effect of process delay was investigated by setting the network rate to 100 *Mbps*, with an inter-packet gap of 1μsecs. The bus transfer rate was set to 50 *MBps* and data length was set to 1024 bytes. The processing delay of the process module was set to 0. 1, 1 and 10 msecs respectively.

Figure 5.17 shows the effect of this process processing delay with respect to protocol processing delay. The measure in this case is not buffer overflow but buffers usage, as the process empties the protocol buffering whenever ready to read.

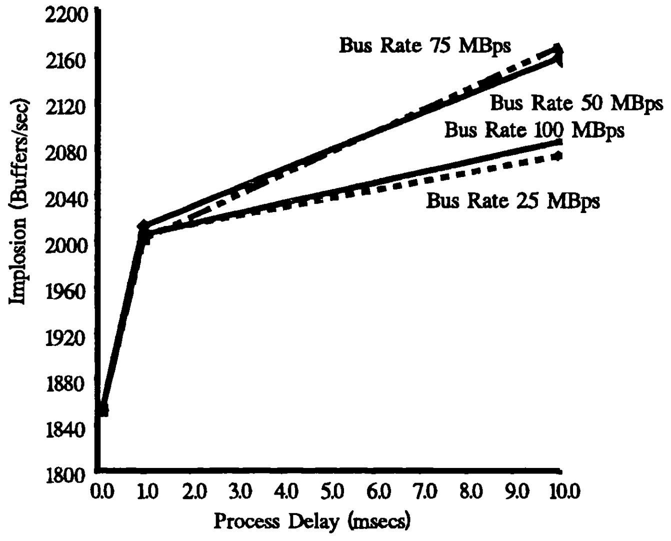


Figure 5.18 Effect of Process Delay during Implosion

Figure 5.18 shows the results obtained for a protocol processing delay of 0. 1 msecs by showing the buffer rate against the process processing delay. Because here the protocol buffering was overflowing, figure 5.12, the expectation was that the process would empty the entire buffer. However, the results indicate that this is not the case, and that far from evening the buffer usage, the process increases buffer overflow.

Initial conclusions from these results indicate that for network rates of around 100 *Mbps* the main problem with multicast is in the protocol processing delay of the host, which relates to the protocol design and the processor capabilities. For all of the results, a major problem is the process, which forces implosion in the protocol level due to the time required by the process to read any buffered packets.

Because of the extensive network buffer overrun for a network transfer rate of 1 *Gbps*, resulting in bus saturation, no useful results were gathered. All results are tabulated in appendix A.

5.4 Data Analysis

The rate at which buffering is used is not directly useful. In order to enable these results to be applicable in determining the actual requirements for the buffering of multicast replies, some measure related to the number of these replies is required. By dividing the rate of buffer overflow in buffers per second by the rate of packet input in *packets per second*, a number between 0 and 1 results, which is in units of buffer per packet, equation 5.1.

$$\mathit{Implosion\ Index} = \frac{\mathit{Buffer\ Overflow}}{\mathit{Packets\ per\ second}} \qquad 5.1$$

A value of 0 indicates that there is no buffer overflow. This number describes the number of buffers that are required to receive a certain number of replies. By multiplying the expected number of replies by this number the number of buffers needed to prevent overflow may be calculated. The number of replies expected is in turn related to the number of recipients, as already described. Therefore, this new measure of implosion, an *index* of implosion may be used to predict the number of buffers required by a host given the architecture of the network and host. Because these latter will be relatively fixed, certainly for the host architecture, the value may be used by that host to configure its internal buffering so that buffer overflow does not occur.

The rate of packet generation depends on the network transfer rate, the inter-packet gap, and the packet size. The packet transfer time is calculated from the bit rate of the network, equation 5.2.

$$\mathit{Packets\ per\ second} = \frac{\mathit{Packet\ Length}}{\mathit{Transfer\ Rate}} + \mathit{Inter\text{-}Packet\ Gap} \qquad 5.2$$

These values may now be used to predict the number of buffers required to buffer a number of replies. For example, if the number of replies is expected to be 20, that is the number of recipients is 20 assuming a positively acknowledged protocol, then a host which has an implosion index of 0.5 for the protocol processing, and an implosion index of 0 for the network buffering will require 10 buffers. However, the experimental results indicate that this is in addition to the minimum requirement of 2 buffers, which results in a need for 12 buffers to avoid implosion. Therefore, the implosion index may be applied using the following:

$$\mathit{Number\ of\ Buffers} = \mathit{Implosion\ Index} \times \mathit{Number\ of\ replies} + 2 \qquad 5.3$$

From this equation it is seen that in conditions of no implosion a minimum of 2 buffers are required. This was based on the observed behaviour of the simulated system under no implosion conditions, a steady state of between one and two buffers being maintained. If the arrival rate of packets was such that each packet was processed before the arrival of the next packet, then only one buffer was required. A value of two buffers as representing the maximum required with no implosion may be inferred by observing that when a buffer size of three is required, then there is always be a packet requiring processing.

5.5 Summary of Results

The results for testing with a network rate of 10 *Mbps* indicates that the processing rate of many of the contemporary workstation processors is more than sufficient from an implosion aspect, exhibiting implosion only for small packets. A major cause of implosion here would be the service time of the application, where the application has to process each reply, and therefore delay the reading of packets from the protocol level. Another factor that may affect the actual number of buffers required is the amount of delay used to optimize protocol processing. In order to reduce the number of context switches required to execute a protocol, which is normally driven both by packet events and timers, the processing of a packet may be delayed in order to give extra time for more packets to arrive, so that a number of packets may be processed in a block. The simulation modules described in the previous chapter ignored this effect and dealt with the capabilities of the host to process a packet on arrival, so that whenever implosion occurred at the protocol level it meant that the entire capacity of the notional processor was occupied protocol processing. Therefore, the results for the protocol level underestimate implosion in a real host, however, these experiments assumed a continuous stream of packets, whereas in reality the number of packets expected would be finite. As these values are intended to indicate the buffer requirements of a finite multicast, then this aspect of the model is not considered significant.

Testing with the network rate set to 100 *Mbps* showed that implosion occurred at the network buffer when the bus transfer rate was set to 10 *MBps*, with the inter-packet gap being less than 10μsecs, figure 5.9. The more interesting results here were those relating to the protocol processing delays which indicated that a protocol processing delay of 0. 1 milliseconds resulted in buffer overflow, figure 5.12. Further tests at this network rate, which investigated the effect of packet length on the implosion characteristics, indicated that as the packet length decreased the implosion increased, with the degree of implosion, figures 5.15 and 5.16, and therefore the implosion index also increasing with decreasing inter-packet gap. These results show that current bus transfer rates are able to handle these network rates, and that the protocol processing will be a problem for many current hosts on networks with this and larger transfer rates.

Testing with the network rate set to 1 *Gbps* showed that implosion occurred for all bus transfer rates, requiring some form of backoff by the network to avoid implosion. The reduction in implosion with packet size, figure 5.11, shows that acknowledgements are more implosion resistant than data replies at the network buffer. This is due to the effect of the packet header, which is stripped at the network buffer, and the inter-packet gap, the combined delay being greater than the delay in transferring the packet across the bus. The reduction in implosion with increasing packet size is due again to the combined effect of inter-packet gap and packet header. However, as above, the implosion is merely deferred to the protocol. Results obtained when the bus was saturated, figure 5.14, do not follow the expected pattern. Initially, the implementation of the model was suspected, but no obvious fault could be found. The nature of the simulation system, being event driven seems a likely source of the problem, with events being lost resulting in packets not being processed by the protocol.

The effect of packet length on implosion at the network buffer showed that for small packets no implosion occurred. This was due to the time taken to transfer the packet over the bus being shorter than the inter-packet gap. The curve after this point may be explained by observing that as the length of the packet increased, the effect of the inter-packet gap is progressively reduced so that eventually the implosion rate would converge to a value reflecting the disparity between the network and bus transfer rates, in effect forming a performance measure.

By taking a range of inter-packet gaps it is hoped that these results may be applicable to a wide range of network architectures. The inter-packet gap was constant for each experiment, which may be compared to a token based network architecture where hosts hold the token for a fixed time period before passing the token to the next host.

The implosion index described above is not only applicable to positively acknowledged and transactional multicast. It has already been described how flow control is used in unicast to reduce buffer overrun at recipients. The implosion index may be used here as well, as implosion occurs when an originator transmits many packets faster than a recipient can accept them. By using the implosion index for a recipient as the main flow control measure, the need for flow control may be reduced, as the behaviour of the recipient may be predicted. The only information required would be the implosion index of the recipient, both for the network buffer and protocol processing with the number of available buffers.

In general, the focus for any work to avoid implosion should centre on the protocol processing and application processing area, as it is here that implosion is most evident, and also where the design of a protocol has the most impact.

The Ethernet experiments show that there is a finite limit on the number of recipients of a multicast that may reside on a single network segment, figure 5.2b. This limit increases with decreasing packet size, but remains less than the maximum number of hosts which may be supported. This maximum may be increased by delaying replies, a delay in the same order of magnitude as the completion time being required. The effect of a multicast on a bridged Ethernet may be inferred by assuming that the bridge is just another source of replies, which it is constrained to transmit in sequence, so that only one reply may be transmitted at a time. It would be expected that the bridge buffer replies, so that replies could be lost by buffer overrun at the bridge, which may be predicted using that hosts implosion index. If the number of replying hosts plus bridges is less than the limit for that packet size then the results indicate that there will be no packets dropped, although the actual limit may be less as there will be more packets being transmitted from the bridges as packets are transmitted from them.

The behaviour of a dedicated network co-processor may be inferred from these results by considering the bus transfer delay as processing delay by the co-processor. Table 5.2 shows this relationship for a number of packet lengths.

Bus Transfer Rate	**Delay in transferring *n* bytes**			
(MBps)	**128**	**512**	**1024**	**4096**
10	1.3×10^{-5}	5.1×10^{-5}	1.02×10^{-4}	4.1×10^{-4}
25	5.1×10^{-6}	2.0×10^{-5}	4.1×10^{-5}	1.6×10^{-4}
50	2.6×10^{-6}	1.0×10^{-5}	2.0×10^{-5}	8.2×10^{-5}
75	1.7×10^{-6}	6.8×10^{-6}	1.4×10^{-5}	5.5×10^{-5}
100	1.3×10^{-6}	5.1×10^{-6}	1.0×10^{-5}	4.1×10^{-5}

Table 5.2 Relationship of bus transfer rate to processing speed

These values may then be used with the graphs above to show the implosion which occurs with these values as processing delays. The use of a co-processor for protocol processing off-loads much of the processing required, especially for acknowledgements. Co-processors typically have a quite large, but fixed buffer space available for use by packets so that if buffer overrun occurs there is no recourse to the operating system to allocate more memory for buffers.

5.6 Summary

Some experiments were carried out to investigate potential implosion, both on a simulated network and on a simulated host. The simulations were intended to show that there may be implosion effects under certain conditions. The parameters which induced implosion were then used to develop a measure of implosion for these configurations.

The results of testing a simulated Ethernet showed that the network architecture caused packets to be dropped because of excessive collisions. The number of replying hosts which exhibited this behaviour varied both with the size of the reply and the length of the Ethernet.

Experiments on a simulated host indicated that implosion was strongly affected by the packet size as well as the network architecture. Data analysis was employed to better quantify implosion, resulting in an *implosion index*, with values ranging from 0 to 1, which may be used to indicate the severity of implosion suffered by a host, but also as a means of determining buffer requirements given a number of replies. The implosion index provides a more useful way of looking at implosion, as it allows an estimation of buffer requirements to be made, and also to predict the number of replies that may be received given a number of buffers, which in turn may allow the maximum group size that may be supported to be determined.

Chapter 6
Designing for Scalability

With implosion identified as a factor in the scalability of positively acknowledged and transactional protocols, and a measure of such implosion derived, a number of techniques to decrease the effects of implosion are described. The implosion index described in the previous chapter is an indication of the severity of implosion. That implosion was found to have a packet length dependency, which in turn indicates that implosion is more of a problem for acknowledgements and small packets than for larger replies.

From the previous discussion of the congestion problem, two ways of reducing implosion are to reduce the number of replies, or to decrease the rate at which replies arrive at the host, that is increase the packet dispersion by artificial means. Both of these methods are discussed.

6.1 Designing for Scale

Simulation indicated that it is the processing speed of hosts that forms the most significant limits on the scalability of multicast, especially on the faster network architecture's now becoming more widespread. Given a particular host hardware, the only areas that are accessible to change are the protocol, operating system and applications. The main areas of these which have an impact on implosion, from a purely processing time viewpoint, are the buffer handling performed by the operating system, the copying of packets from operating system to process and the need for ensuring data integrity. While these issues are largely performance related, and therefore generally applicable, they are included as forming the first step in designing a multicast protocol for scalability, the UNIX operating system [Le89a,St90a] used as a reference.

The efficiency of a protocol's internal operation can be separated into those parts which are operating system dependent, such as buffer allocation, packet copying, and those which are protocol dependent, such as the checksumming of packets and searching for state information. Reducing the processing time of a protocol requires that these elements be investigated with the aim of reducing their respective processing delays.

One scalability aspect of multicast concerns the amount of information about recipients that an originator requires for the correct operation of a given protocol design. This is important for two reasons. The first is that the originator must store and access this information, which takes both time and possibly valuable memory. Secondly, the originator must be able to reference, update and maintain this information, consuming processor time. Although these are group management issues, many of the protocol designs discussed earlier require the searching of information on a per packet basis, most notably positively acknowledged and transactional designs.

The amount of information held may no longer be a particular issue, as reducing memory prices and the subsequent increase in the amount of memory now seen on hosts have to some extent reduced the problem. However, memory used for such information is not available for use by applications, and some form of compaction may be possible. The technique used to implement a protocol may be of help here, as some this information may not be necessary for the operation of the protocol, being used for group management purposes rather than to enable the operation of the protocol, and therefore may be stored separately, possibly using virtual memory where the potentially slower access time is less of a problem.

The referencing of state information may consume a large part of a protocol's overall processing delay, especially for multicast. In a positively acknowledged design, each acknowledgement results in the accessing of that recipient's information. By definition, every recipient replies to a multicast, requiring the originator to search for state information relating to that recipient for each reply. Therefore, any method to reduce this search time would be beneficial. The problems of referencing protocol information was investigated by Clark, Jacobsen et. al. [Cl89a] with reference to the performance expectations of the *TCP*. Among the recommendations made were optimisations such as using a reference to the last referenced protocol information block, which is the first tested when a reply is received. Because the *TCP* tends to transmit many packets at a time, this optimization reduced searching time substantially. Multicast does not necessarily show this kind of behaviour, but the use of a faster method for referencing information, such as hashing [Ja89a,Ja89b], would benefit multicast for large numbers of recipients.

Copying is used by UNIX, and other operating systems, to ensure that the operating systems internal structures are not accessible by processes. Without the copying of data between application and operating system proper it would be possible for the application to modify a buffer that is already being transmitted. There are two main areas where packet copying may occur. The first is as above. The second is copying from the bus to main memory. The simulation assumed that the bus was capable of Direct Memory Access (*DMA*) transfers, where the network device is able to copy the packet directly to main memory without the intervention of the main processor. The absence of *DMA* would effectively increase the processing time for each packet as the main processor would have to carry out copying. However copying is done, there must be some area of memory into which the packet may be copied, which is most often located in the operating system proper. As packets are of variable length buffering can result in either inefficient memory usage or slower buffer handling.

UNIX uses small, fixed size blocks of memory which are then chained together to contain each packet, thereby minimizing the amount of unused memory. However, the management of these buffers requires a relatively large overhead, especially when dealing with headers, requiring that after header processing the data portion of the packet be adjusted so that the data starts at the beginning of a memory block. In addition, *DMA* must employ scatter/gather techniques as the data may be copied into non-contiguous blocks of memory. The *V-Kernel* [Ch84b] employed a Uniform Input Output [Ch87a] system uses blocks of memory large enough for the largest packet for each received packet regardless of size. Although memory wastage is higher, the scheme is more time efficient, as much manipulation of buffers is removed. The most efficient method would be to allow packets to be copied directly into application memory, the packet being processed in the usual way before the application is in effect handed back the buffer. Among the problems of this is the need to protect the contents of a buffer during any copying.

If a host supports many group members, then an efficient Inter-Process Communication method would reduce processing time. If the packet must be copied to each member then processing delay would increase in proportion to the number of members. Using shared memory between these group members would allow only one copy to be made, which would be removed when each member had referenced it. If the packet was to be manipulated, rather than just read then a copy of the packet must be made before hand and made private to that member. The shared memory area would be created by the first group member on that host.

The checksumming of packets to verify the data is a common method of ensuring data integrity. A checksum is a value that is calculated using a formula based on the data contained in a packet. The time taken to calculate a checksum is therefore proportional to the length of a packet, although checksumming may take place in parallel with data reception if supported. Another operation that is dependent on the length of a packet is copying. One method for reducing the overall time consumed by these two processes would be to combine copying and checksumming, so that the overall time decreased. Of course, some means of rejecting a copy if the checksum were faulty would be required.

The role of the application, and more specifically the interface to the application must not be ignored. The results indicate that the time taken for an application to respond to the arrival of messages greatly affects the implosion at the protocol level. A number of unknowns exist in such a situation, depending on the number of runnable processes in time-shared systems, the priority of such and where process switching takes place. For example, the UNIX operating system switches whenever input or output occurs, which is precisely where implosion is most sensitive. It is here that the design of an interface has an impact. If the application has to read individually each message, then implosion is more likely to occur than if a single read gathers all replies.

The simulation ignored overrun at the process level, because implosion does not in fact occur there. What does happen is that the inability of a process to read more packets causes overflow in the protocol level, as shown in the results. Of course the application only affects the protocol in this way if the multicast is transactional, or the multicast protocol itself is part of the application. An application may be able to allocate a large number of buffers, using virtual memory, but not all of these buffers may be in real memory at a time. Therefore, more time is required to swap these buffers back into memory, increasing overrun at the protocol. The number of packets that are of value to the application may be used to decrease this problem, by rejected any messages that do not match a certain criterion. If the voting algorithm can be implemented within the operating system, then the buffer overrun may be reduced. The ability of UNIX *STREAMS* [T89a] to push different protocol modules onto a stack allows modules to be used which may selectively filter messages before they are copied into the application.

So far the assumption has been that protocol processing takes place in main memory. The use of special hardware, such as a co-processor dedicated to network processing, has not been considered. Co-processor boards typically process packets up to the transport level, so that only data is actually transferred over the bus. The results in Chapter 4 indicated that the speed of the bus often outstripped network speeds, and that the limiting factor was on protocol processing speed. This would indicate that most advantage would be gained in using a very high performance processor to protocol process, which would allow for less processing by the main processor if data has to be transferred.

If, after exhausting the possibilities above, implosion remains a problem, especially if the implosion occurs below the bus, then the technique used for data transfer should be considered. Two basic techniques were considered with a view to their scalability. The first was considered because of the techniques inherent reduction in reply traffic. The use of negative acknowledgements, because replies are only generated when a packet is missed by a recipient, offers an immediate way of reducing implosion. However the technique also has a number of inherent drawbacks that have to be addressed. Positive acknowledgements, where a reply is transmitted confirming the reception of a packet, are considered implosion prone due to the synchronizing effect of a multicast. The technique however has a number of features considered necessary for many applications. Indeed, transactional protocols, where data forms the reply, are being increasingly utilized in distributed applications. The technique used to implement a protocol design has a potentially powerful impact on the scalability of a protocol and should be considered with care. A number of factors should be considered when assessing technique, an important one being that the design should not penalize small groups, even if the design is intended for large scale use.

The number of groups supported by a single host, and the number of groups that can operate simultaneously are also scalability issues. The number of groups supported by a host is related to the problem of address filtering. In order to reduce the number of unnecessary multicasts received by a host it is preferable for address filtering to take place as low as possible in the protocol hierarchy. Each address must be stored and referenced, which may be beyond the capabilities of the network device, in which case a special *all-multicast* address may be employed. Again, the time taken to filter on a multicast address may be significant.

6.2 Negative Acknowledgement

It was stated earlier that the use of negative acknowledgements for multicasting is inherently scalable. However, it was also stated that the technique possessed a number of problems that restricted its use for many applications. A number of methods may be employed to overcome, to some extent, these problems, although these methods may reduce the scalability potential of the basic method. The technique is scalable primarily due to the minimum use of replies from recipients to originator, thereby avoiding buffer overrun directly. On the other hand most of the problems of the negative acknowledgement technique are due to this lack of replies. Another feature of the technique that aids in scalability is the lack of any need by the protocol for any information about the recipients as all information necessary for retransmitting a missed packet is contained in the *NACK* packet transmitted in response to the detected loss of a packet. Coupled with the reduced number of replies, for non-transactional use, the processing delay of this technique may be considered small compared to other methods.

Reliability is one of the primary design parameters for protocols. Using negative acknowledgements are inherently unreliable because an originator cannot distinguish between a recipient missing a packet and successfully receiving it, the two cases having the same effect. A protocol using this technique can only be described as offering a *degree* of reliability. In a negatively acknowledged protocol it is the recipient that has responsibility for reliability, a recipient transmitting a negative acknowledgement packet to the originator when it detects a missed data packet. The detection of missed packets is often based on observing the sequence numbers of received packets, a gap in sequence numbers indicating a missed packet. When a gap is detected, one or more negative acknowledgements are generated and transmitted to the originator of the missed packet. Two problems are immediately apparent. The first is that for this scheme to operate, there must be a subsequent received packet to trigger gap detection, the so called "last packet problem". Secondly, the detection of a missed packet by a recipient must enable the recipient to reconstruct the originator and the identity of the missed packet for a negative acknowledgement to be correctly addressed.

A solution to both of these problems is for the originator to multicast either a copy of the last packet, or a dummy packet containing the next sequence number which would be transmitted when data is to be transmitted, the reception of either type of packet resulting in the detection of any previously missed packets. These extra packets could be repeatedly transmitted until either a new data packet is to be transmitted or the transmission has been repeated a sufficient number of times to satisfy some reliability criteria.

Instead of transmitting or retransmitting these extra packets, a timer could be used by recipients that is set on arrival of each packet. If the timer expires, then it would indicate that time has passed since the last packet was received and forces the recipient to check for missing packets. If the timed gap was set to a value which reflects the time taken to receive a packet, that is the time between the arrivals, then expiry would indicate a missed packet. By including information about the number of packets being transmitted in each packet, then checking for missed packets will also detect when a last packet has been received. The number of packets being transmitted would be available if the unit of transmission was the message, rather than the stream. By setting the timer such that the delay is calculated based on the number of packets in a message, so that expiry of the timer would occur after the last packet of the message was expected would reduce the number of timer operations. However, this scheme does not cater for messages which are small, where the loss of a packet ensures that the presence of the message is not even detected. Also, the calculation of the timer would require information about the expected rate of transmission, which is also influenced by network activity. As the main cause for packet loss is congestion, here at recipients, it is with this that the discussion turns.

One factor affecting the number of missed packets is congestion, leading to buffer overrun at recipients, as opposed to implosion and buffer overrun at the originator. A popular method of controlling this is to use a token, which allows an originator to transmit one or more packets to recipients. As the aim of using the negative acknowledgement technique is to reduce necessary replies the token must not be required by an originator to transmit, but rather as a means of modifying the number of packets that may be transmitted as transmission progresses. This is contrasted with the same methods use in positively acknowledged designs, where the token is used before transmission starts. Another method would be to control the rate at which packets are generated, the aim being to settle for a rate which is maintainable by every recipient. The rate may be based on some observation of patterns to missed packets, as in the *VMTP* [Ch86a] or simply using a delay system which operates for every missed packet. Again to reduce the number of replies, recipients must communicate with the originator only to change the transmission rate, assuming a default value for initial transmission. Reducing lost packets using these methods does not preclude the need for re-transmission in the event that some packets are lost.

The reliability mechanism of the negative acknowledgement technique relies on recipients being able to recover missed packets from the originator of those packets. This requires that the originator retain packets until it can be assured that recipients have had a number of opportunities to detect and recover any missed packets. The recovery of buffer space at the originator thus forms one of the problems of using negative acknowledgements. If a packet is discarded too early by the originator then any subsequent negative acknowledgement cannot, in most cases, be satisfied. In certain circumstances, for example if the data is stored on a non-volatile medium, such as a hard disk, then some higher level protocol could be used to retransmit missing sections, this being dependent on the usage of the protocol. It should be noted that because of the lack of replies buffer recovery is a based on rules of thumb rather than any precise method. A number of methods may be used to manage the buffering of packets at the originator.

The simplest would be to retain packets until forced to discard due to more packets being transmitted. The reliability of such a scheme is highly dependent on the generation rate of packets, and is more suited to a low generation rate environment. A better method would be to guarantee that each packet is retained for at least a fixed amount of time, this time being calculated from the number of opportunities granted to recipients of receiving each packet, any new packets being delayed until buffer space is free. A more adaptable method is to use information which is transmitted, either periodically in the absence of traffic or as part of normal protocol traffic, by recipients, so that the originator can track the last received packet for each recipient, discarded any packets which are older than the oldest seen. One of the problems with this approach is the requirement that the originator receive packets from all potential recipients, which may result in implosion, but also that the amount of additional data carried in each packet may be significant, requiring in the general case the address of the originator and the sequence number of the last packet from it, for each originator. Also, the use of broadcast is implied for every packet, as the information has to be directed to every originator.

A different approach to this problem is to use *saturation* in conjunction with negative acknowledgements as a means of increasing the degree of reliability offered by the technique as described by Jones, Sorensen and Wilbur [Jo91a]. Saturation means that many copies of a message are transmitted, thereby increasing the probability that a recipient receives at least part of the message. The main aim of using saturation was to increase the probability that a small message, one containing one or more packets had a higher probability of being detected, detection of only one packet of a message assuring that the message would eventually be received through the use of negative acknowledgements. Using saturation directly would result in many redundant packets being transmitted, especially if the loss rate for the environment is low. Therefore saturation was concentrated whenever the message to be transmitted was shorter than a *threshold* size. When this was so, multiple copies of selected packets were transmitted to increase the probability that at least one packet were received by every recipient. Otherwise, the packets of the message were multicast individually.

The use of saturation in this way was prompted by the observation that as the size of a group increased, the probability that each group member received a particular multicast reduced, increasing the need for retransmissions. Also, the design assumed that no more messages would follow the current one, so that if the current message was missed, there would be no subsequent message to prompt for it, so that enough information was conveyed in each packet for the whole message to be recovered.

The use of negative acknowledgements for requesting retransmissions is intended to avoid synchronization of replies. Because hosts miss packets entirely randomly and because the number of such missed packets may be assumed low this goal is effectively met. However, if a common mode failure, such as a packet being discarded at an intermediate bridge, then synchronization is introduced. But, only one negative acknowledgement needs to be received for that packet to be retransmitted, satisfying all of them, assuming that retransmission is also multicast.

The efficacy of negative acknowledgements is biased towards the technique's use on low loss low delay networks. Low loss because of the problems of detecting missed last packets, low delay because of the consequent effect on buffer recovery at originators. The use of negative acknowledgements is thus effectively confined to networks with these characteristics, although they can be effectively used in conjunction with positive acknowledgements, typically with the last packet of a message being positively acknowledged.

The requirements of group management information by protocols was introduced earlier. One of the advantages in using negative acknowledgements is the lack of knowledge about recipients required by the originator, implying that group management is not required for these types of protocols. While group management should always be considered when discussing multicast, a negatively acknowledged protocol does not require that information to operate.

Incidentally, the technique is useful if the data is time critical. As will be described later, a positively acknowledged design has a completion time proportional to the number of recipients, simply because each recipient must reply to the multicast. Therefore, if the time needed for gathering these replies is larger than acceptable from a time critical point of view, then the use of negative acknowledgements may offer a more suitable method for operating with time constraints, especially if the network has a low delay characteristic, allowing recipients at least some time to recover missed packets. In summary, negative acknowledgements offer inherent scalability, but lack reliability. For time critical activities, the potentially low completion time is an advantage.

6.3 Positive Acknowledgement and Transactional

The primary advantage in using positive acknowledgements is that the transmission is reliable. The originator has confirmation that a packet has been received by the intended recipient, which benefits buffer storage, because acknowledged packets can now be safely discarded. However, requiring replies from each recipient is a potential source of implosion.

The results described in Chapter 4 indicate that implosion for acknowledgements, which are typically short packets, is a problem. Of course, implosion is not confined to short packets, but the problems are more acute for acknowledgements. This may be fortuitous as acknowledgements may be delayed for a short time, delay being used to reduce synchronization and therefore to some degree reduce implosion. However, acknowledgements are not subject to the flow control restrictions described previously, and therefore require some other means to reduce implosion to manageable levels, that is to prevent buffer overrun.

A positively acknowledged protocol here is considered to require at least that the last packet of a message be acknowledged, with other acknowledgements being transmitted if packets are missed. Flow control is by the use of a token to indicate buffer availability or by rate control. Each recipient must acknowledge at least once per message, which effectively excludes the use of optimistic algorithms, which assume that if a number of recipients acknowledge, the group as a whole is considered to have received the message, using the assumption that any missed packets may be recovered from those recipients that acknowledged the message.

A number of possible approaches may be used to reduce implosion at an originator. Two basic approaches may be taken. The first is to increase the time between replies so that overrun does not occur, that is increasing dispersion. The second is to ensure that the number of replies directed to any one host does not result in implosion at that host, as there is sufficient buffering available to ensure that buffer overrun does not occur.

One way of reducing implosion is to accept that implosion occurs, requiring retransmission, but that the retransmission is replied to only by those recipients from which the originator has not yet received an acknowledgement. After many retransmissions, the number of recipients would fall to a level where no further implosion occurs. This method requires that information about the recipient replies that were received is passed to the recipients on retransmission. For large groups this method may require an excessive number of retransmissions, as well as requiring a large amount of information to be passed to the recipients. Using the most efficient method of representing recipients, using one bit per recipient, the number of bits is equal to the number of recipients. To be flexible, the number of bits employed needs to be varied depending on the number of recipients, as well as group management to ensure that each recipient is allocated a unique position in the bit table, as well as ensuring that the bit table does not become stale.

In a closed group environment, where communication is between group members and crucially the originator is a member of the group, acknowledgements may be piggybacked onto data transmissions, reducing the number of acknowledgements transmitted. In order to increase the likelihood that a data packet can be used in this way, acknowledgements may be delayed for a period. In fact, implosion in a closed group is probably less likely, as the pattern of communication would tend to be less synchronized than for service location, for example.

Distributing Replies in Time

The host model developed in Chapter 3 identifies a number of factors affecting implosion. Of these, for a given host, only the inter-packet gap of the network may be used to decrease implosion. Therefore, an obvious method for reducing implosion would be to arrange replies such that each reply is preceded by a gap sufficiently large to negate or reduce any implosion. One of the drawbacks to delaying transmission is that the network is no longer being utilized fully, increasing completion times and potentially delaying other non-multicast traffic co-existing with multicast replies.

If the architecture of the network does not already supply a sufficient gap between replies then the recipients must artificially increase the gap by delaying the transmission of replies subject to some distributed algorithm. The most obvious method of arranging the replies is to place each reply in a fixed slot, the size of which reflects the implosion characteristics of the originator, which may be determined using that hosts implosion indices, biased for the expected size of the reply, figure 6.1.

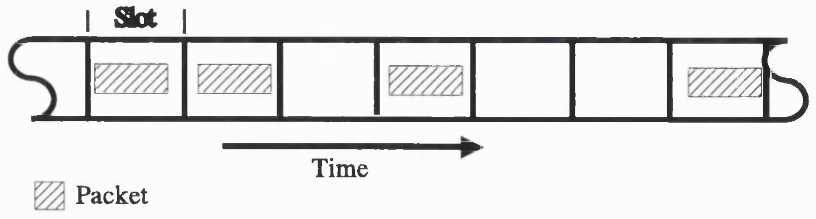


Figure 6.1 Time Slot Scheme

The first problem which must be addressed concerns the allocation of these slots among the recipients. For the scheme to operate each recipient must firstly know which slot of all potential slots it has the exclusive use of, and secondly when to transmit in order to fill its allocated slot. Group management finds a role here, as the group management operations for joining, leaving and failure detection can be used to allocate hosts unoccupied slots, and to recognize free slots for allocation. Additionally, group management can be used to expand or contract the slot system depending on the group size. In order for slot management to be efficient, the number of unused slots should be minimized by rearranging the allocation of slots. However, this would require communication with recipients, as well as some means to calculate the allocation of slots.

As the time slots are not a function of the network the synchronization of these slots, so that each recipient knows when to transmit, must be handled by the recipients. The multicast from the originator may be used, the reception of the multicast forming the synchronization signal. However, recipients will almost certainly not receive the multicast at the same time, due to the propagation delay imposed by the network architecture. In addition, external traffic, not being a party to the slot scheme, would tend to disrupt the smooth running of the scheme, delaying recipients past their allocated slots and into others. One method of reducing the synchronization problem would be to use a slot of a size that would cater for propagation delay, and accept a certain amount of variation in the arrival times, keeping this variation within the originator's implosion limit. Other methods could listen for reply traffic, and base its synchronizing signal on the first reply seen.

If a host can sustain a certain amount of implosion then using a random slot allocation scheme may offer a simpler means of solving the problem. While a random method may produce implosion due to slots being picked by many recipients, the implosion would tend to be smaller, allowing normal buffering to operate as intended. The technique would be more robust to failures, the only adjustment required being new random generator values. Indeed, it would only require adjustment of the slot time and random generator values to increase or decrease observed implosion.

The size of slots would have to be calculated based on the inter-packet gap required and the size of the expected packet. While this would favour acknowledgements, which are typically of fixed size, the scheme would be less favourable for variable sized, or even multiple packet replies. The former can be solved by using the maximum packet size, and adjusting the inter-packet gap accordingly. Using the slot scheme for possible multiple packet replies is more problematic. Either each recipient reserves a number of slots for itself, or the slot scheme repeats for each packet of the reply.

A similar scheme to using time slots for replies is to use a token, which is passed from recipient to recipient, to grant each recipient access to the media. By controlling the passing rate of the token, a slot system is enforced. However, the use of a token here requires some means of guaranteeing the reliability of token passing, or at least some means of reconstructing the token if lost.

A more robust method for replying would be to impose a randomly generated delay on to replies, using the network to resolve any conflicts. Here there is no fixed time slot to which a recipient is bound, and therefore no need to manage slot allocation. The values used to drive the random delay generator would then just have to be multicast to the group when required.

Distributing Replies in Space

Distributing replies by space rather than time means restricting the number of replies which any one recipient has to receive. The basic case of this is where the group management of a protocol restricts the group size to prevent buffer overrun. Such a method is not scalable, as the number of group members is restricted. However, the scheme can be extended by using a hierarchy of group members, where the group as a whole is split into smaller, more manageable sub-groups. Each sub-group is then headed by a single group member, just as above, but here that group member is itself a member of a sub-group. While this can also be said to be time distributed, in view of the extra delay imposed by forwarding, the major gain is in the distribution of the replies. The advantage of this approach is that no single host is expected to receive and process all of the messages individually. By filtering replies at each level much of the traffic generated can be reduced. In addition, the number of recipients supported by intermediate recipients may be tailored to that recipients buffer configuration.

One reason that distribution in space is considered of more importance than that of time is by observing the relative speeds of networks and processors. In a time distributed system the bottleneck will be the ability of the host to receive and process all of the messages within a certain time, whereas the bottleneck in the space distributed system is the network as each host will only see and process a part of the returning messages, and as network speeds have increased far more in recent years than processor speeds, the time distributed system is expected to be unable to use the potential offered by the new network architectures. In addition, as there is no delaying of traffic, the full capabilities of the network may be utilized.

The hierarchical structure developed here is intended to outline not so much a final solution but a philosophy that can be applied to a number of protocol designs. The tree structure is developed as hosts join a group, with a joining host being allocated by management functions to maintain a tree structure, which provides a degree of load balancing, each host having an optimal number of supported child hosts. A *child* host is one that possesses a *parent*, to which replies are directed. A *sub-group* is a collection of hosts which share the same parent, and are members of the same multicast group. A number of sub-groups whose parents are a sub-group make a *level*. The tree structure is not *rooted* by a single host, but by a level. The levels are numbered from zero, with the *root group* residing at level zero. As more hosts are added to the group each level fills until the maximum allowable number of each level has been reached, where the next level is started. The placement of the joining hosts is discussed with reference to the group management requirements. The size of each sub-group may be determined by the implosion indices of the parent.

The structure described above has three types of group member. The root group members reside at the top of the tree and are the entry point for the originators of multicasts. These originators are, in effect, parents of the root level, which implies that only root level members communicate with originators. There is no limit on the number of originators supported. A *leaf member* has no children and therefore only communicates with its parent. An *intermediate member* is neither a leaf nor a root member, but has both children and a single parent. The intermediate member has to gather replies from its children and forward any information to its parent. Thus for leaf and intermediate members each sub-group has a single parent, which it is allocated by the management operations discussed below. The problems of the *redistribution* of group members when hosts leave the structure or fail are properly a group management issue.

The propagation of messages from an originator to the group can have two distinct forms. The first is referred to as **strict hierarchical forwarding**, figure 6.2

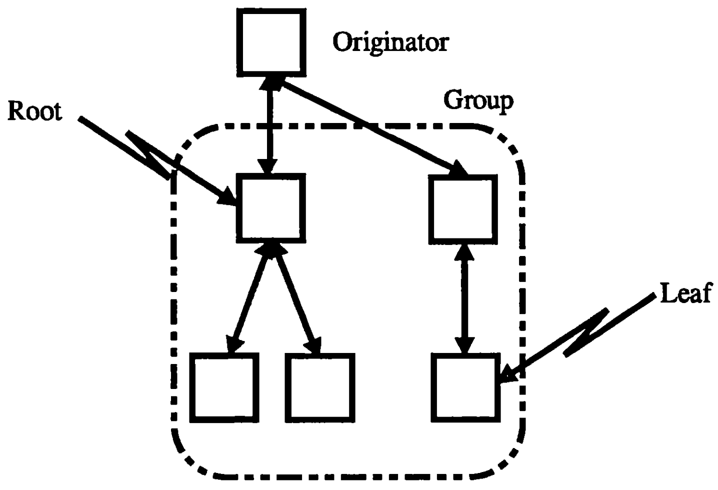


Figure 6.2 Strict Hierarchical Forwarding

where messages are actively forwarded by the intermediate group members to their immediate children, which in turn forward the message to their children. Replies flow back to each parent in a similar fashion. One restriction on such a scheme is that each message must be unicast to every host as the parent must transmit to only its children, preventing the use of multicast or broadcast. If a special addressing scheme is used which supports sub-group addressing, then multicast can be used. Suitable arrangements of sub-groups with parents may be made, so that a parent and its immediate children are resident on the same network segment. This is beneficial from a traffic point of view, but also allows strict hierarchical forwarding to be applied using broadcast, simply by denying broadcast, or multicast forwarding by bridges.

Loose hierarchical forwarding, figure 6.3, is of more interest as

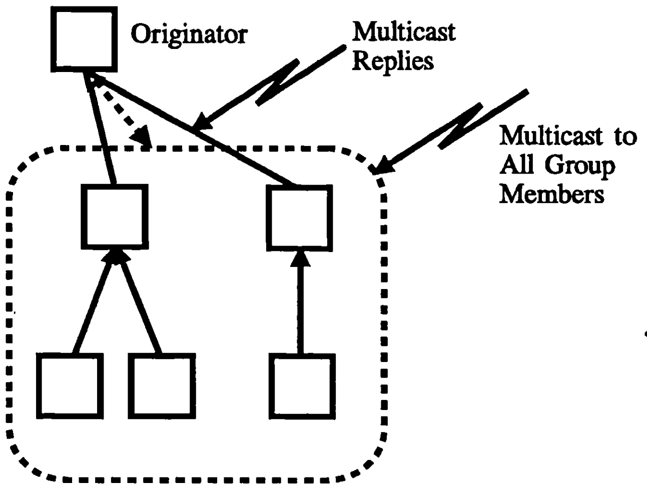


Figure 6.3 Loose Hierarchical Forwarding

it allows a broadcast capable network to be used directly. In loose hierarchical forwarding the message is multicast by the originator to the group, with replies from each recipient being passed to its *parent* rather than the originator. In this way replies are distributed among the set of parents, reducing implosion. Using such a scheme then introduces additional options with regard to the retransmission policy observed by each host.

In using the loose hierarchical model the retransmission strategy used by the protocol can take two forms. Either all retransmissions originate from the originator, or retransmissions are delegated to any parent which has received the missed packet, thus reducing the load on the originator. The latter is of preference, as it allows the originator to complete earlier than may be the case if it has responsibility to retransmit. In addition, intermediate hosts may be able to retransmit in parallel.

If replies are propagated back to the originator then parents delay their own replies until replies have been received by all of its children. The originator will therefore only receive a reply if an entire branch of the tree has received the message. Because the originator has no knowledge of which recipients actually missed packets, the missed packets must be re-multicast, resulting in duplicates being received by many recipients. If only one branch failed to acknowledge, then it may be possible to use strict hierarchical forwarding over that branch alone.

If the onus of retransmission is delegated to the group, then parents have the responsibility of retransmitting missed packets. This assumes that a parent has received the missed packets itself. What can be assumed is that eventually it will receive those packets, at which time it in turn may forward missing packets. The originator may complete when it has received the appropriate replies from the level zero hosts, although the group as a whole may not at that time have completed. Retransmission may be unicast, if the number of missed packets is confined to a small number of branches, or multicast. If the latter is used, then the retransmission would be receivable by all group members. The performance penalty for this may be acceptable if infrequent.

So far the assumption has been that replies are acknowledgements that are received by parents, and destroyed there. However, if the reply is data then these data messages must be forwarded from the parent to its parent. The concentration of replies at a host offers a number of methods for pruning the number of returns. For example, if only one reply is required then other replies may be discarded at intermediate hosts. A majority can also be treated in this way. Replies could be forwarded as they arrive at the parent. A problem with this is that the number of packets forwarded by intermediate hosts to their parents is related to the number of children possessed by each intermediate host. This may cause an implosion at its parent.

A solution is found in the use of flow control by parents. Using a token to indicate buffer availability to children may also be used to prevent children replying, acting as a block on transmission. Access to parent buffering can then be granted in a round robin fashion, so that each child is in effect polled for its data. One problem remains. After collecting these replies, the parent may have a large number of messages stored for forwarding. The problems of storing these messages may require either that some form of virtual buffering be used, or that messages are forwarded whenever the available buffering is filled. Which ever method is used, in effect the replies are blocked forming a larger, multi-part message. The same technique can be used to gather these messages into ever larger messages as the replies move up the tree, subject to any filtering described previously. Because these messages will become ever larger as the replies filter up to the originators, the hierarchical structure will become in some way limited in depth, that limit being determined by the sizes of replies and the number of recipients replying as well as the proportion of messages that are forwarded by intermediates.

Using this structure in an open group model, where originators communicate with the group using some form of access operation is of interest because of the reduction in information transfer which can be achieved. Because originators only communicate with, and therefore have knowledge of, the level zero members, the amount of information required is reduced. Also, the amount of information stored at each group member is reduced, the maximum being at intermediate hosts. The management information required by the protocol is extensively discussed below.

The maintenance of the tree structure is a function of group management, the successful operation of a hierarchical scheme requiring that group management operations be used to maintain the structure. The next section describes the functionality and requirements needed by these management operations.

Managing the Hierarchy

Chapter 1 described a number of basic management operations which were considered to be necessary for multicast protocols. In this section the implications of a hierarchical structure on the format and usage of these operations is considered.

When discussing group management it is necessary to first define what information is to be stored at each host. This in turn depends on the frequency of each operation that occurs for a group. In the structured approach one of the goals is to reduce the amount of information held by any one group member, which is achieved by distributing the information among each group member in such a way that a recovery from failure can be carried out with the minimum amount of overhead. As the group is by implication open, then there is no gain in having complete information at each group member, as in *ISIS*, as a user will not necessarily be a group member, although it can be. The details of the information required is left until after a discussion of the group operations. In *ISIS* management operations are ordered relative to all other activity, ensuring that the group view is updated in a consistent manner. If no group view is used then some other method is required to ensure that there is no group change during an interaction. This could just take the form of delaying a group management operation until any pending interactions have finished, with colliding management operations being prioritized depending on the management operation.

A create operation in a hierarchical structure is no different to a create in a flat structure as the hierarchy has only one member, and can be implemented in any of the ways discussed previously.

A destroy operation would use the structure to propagate the destroy instruction and would therefore require that the destroy be delayed for each group member until all of its children have acknowledged the destroy operation. The destroy would therefore begin at the leaf hosts and propagate up the tree. The usefulness of a destroy operation has been discussed previously.

The joining of the group is used to place the new member within the hierarchy already established. The placement decision can be based on a number of factors, not least the level at which the new member is inserted. The number of children that can be supported by a parent, based on the implosion resistance of that parent, is used to place a limit on the number of children of the level, with a new level being started when that limit is exceeded. One problem with this approach is in anticipating the implosion resistance of originators, which impacts the size of the root sub-group. The larger this group the more efficient the protocol, because of the possible reduction in the number of levels for a given group size. The create operation often includes a join for the first member. If the joining host is placed at the root level, then a number of actions must take place. Firstly, the other root level hosts must be informed of the new member, and the new member must gather state information about the others. In addition, the new member must also find out about any originators. The originators may subsequently be informed of the group change, or discover the change when required.

When the joining host is placed at another level, then it is allocated a parent, which is informed of its new child by the group management. The child is informed of its parent. The placement decision depends on a number of possible factors, such as the implosion limit of that parent, and possibly load balancing factors, such as the ration of children to parents for that level. A number of optimisations may be made. For example, it would be wasteful to create a new level populated by only one group member. Therefore, a certain amount of leeway should be used when joining a group, allowing a few members to accumulate before creating a new level.

If a host leaves a group for any reason the actions required to maintain the tree structure will vary depending on the leaving hosts location in the tree as well as its manner of leaving and if the tree is required to remain balanced. If a leaf host leaves then the operations required to maintain the tree are minimal, just involving the deletion of state information held for the leaving host. If a non-leaf group member leaves the group then either its children can be redistributed to all of the other hosts at the same level as the leaving host, or one of the children can be elevated to take the leaving hosts place, and become a parent to the other members of the sub-group. The latter is more flexible as well maintaining the degree of distribution of the tree. In addition, if a root level host leaves the group, then the elevation of a child is essential, as redistribution may not be possible.

A more difficult case occurs when a host fails, figure 6.4, requiring failure detection followed by a group leave on behalf of the failed host.

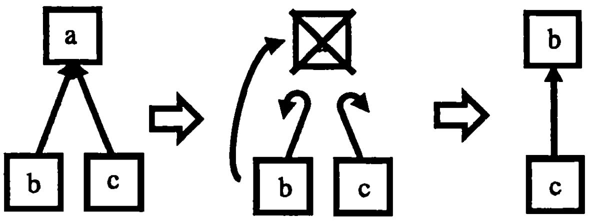


Figure 6.4 Recovery from Failure of Hierarchy

An advantage of the hierarchical approach is that the hierarchy isolates parts of the structure from all other parts, so that the group reformation can be restricted to concerning only a small proportion of the total. Again, the actions to be followed when a host fails depends on the failed host's position within the hierarchy.

If a leaf host fails then its failure is detected by successive timeouts by its parent. If the group management is required to keep track of current group membership then some form of probe will be required, which can either be a periodic transmission from each child to its parent, or a poll by the parent to its children, or a broadcast by a group monitor, which would result in acknowledgements being transmitted up the tree. The probe can be likened to the "stay alive" function implemented in the *TCP*, which uses an out of window packet to force an acknowledgement from the other side. Alternatively, if group consistency is not required then a failure can be detected whenever a message is transmitted by a host to the group.

If the failed host is a non-leaf host, but has a parent, then the parent as the additional responsibility of somehow locating the children of the failed host and informing them of the failure. Once the children have discovered the failure they can then elevate one of their own members to take the failed host's place. Children can discover a failure either through the non-response of its parent when a child requires a retransmission, or by a group leave operation carried out by the parent of the failed host. The latter case is a more reliable method.

If a root member fails, then it has no parent to detect its failure. The parents of the root group are the originators, so that an originator that does not receive a reply from a failed member may then issue a group leave for that member. Because the originator has no knowledge of the failed member's children, the leave must be multicast. An alternative is for the other root group members to detect the failure, issuing a leave group as before. The latter is neater as it does not require intervention by originators, but does then require some means of detecting failure, such as using a periodic multicast which is replied to by root group members.

If a root member fails then one of its children is elevated, with the child also having a number of children, then either the children can be redistributed to the other hosts, or one of the child's children can be elevated to take the child's place, thus retaining consistency.

These observations lead to an idea of the type of information required by each host. Any host that has child members requires that it has a descriptor block for each child, each host on the same level and each parent, where a parent can be either a host one level higher or, in the case of a level zero host, the clients of the group. Leaf nodes require knowledge about each host on the same level and each parent. The logic of this is apparent from the need to elevate hosts. If each host in a sub-group has information about all of the other hosts in the same sub-group, then it is a simple matter to elevate any host, as all of the information is already in the elevated host's possession. Equally, each host must know about its parent(s) and its children. Therefore, the maximum amount of information possessed by any one host is twice the maximum sub-group size plus one or more parents.

One other group operation which must be included is the use operation. A user host wishing to multicast to the group has to locate and set up descriptor blocks for each level zero host. This setting up will involve initial parameter settings and is mainly used for detecting a level zero failure. The setting up could take the form of a group query, which is multicast to the group and forces each host to reply with its own address and possibly the addresses of all of the other group members that that host has knowledge about, providing a cross check of the current group state. All of this information would be filtered as it passes up the tree, until the level zero hosts send their information to the requesting host. One advantage of this approach is that the group is checked for consistency on every group query. However the costs for doing so may be prohibitive, so a special command could be used which is handled only by level zero hosts, which are identified by having no designated parent, to reduce the traffic generated. If no such set-up is desirable, for example in a remote procedure call context, then the state information could be gathered as a result of a straight multicast, with each reply containing the replying address, and possibly a list of n other members of the level zero group, n being less than or equal to the optimum sub-group size.

Communication of group management operations uses the structure already described to gather replies. This is a problem when strict hierarchical forwarding is used, as the structure may not allow forwarding because of failure at a node. Therefore, a true multicast must be used if possible to inform the relevant members.

6.4 Wide Area Networking

The problems, and some solutions, of multicasting in a *WAN* environment have already been discussed. These concerned the routing of packets within a network. The scalability of such system has now to be considered.

The implosion of packets in a *WAN* can be considered from two points of view. Firstly, implosion within a *WAN*, where packets are being received and forwarded internally to the *WAN*. The second is the interface or gateway between *WAN* and *LAN* or *MAN*. In the former case, the transfer rates of the input and output are likely to be similar and slow. Therefore it is likely that any implosion will be affected by traffic rather than an inability of the host to cope with demand. Implosion would also increase the closer to the originator, as the packets will be being funnelled by the routing mechanism, to ever fewer bridges. To offset implosion is the greater range of delays exhibited by a *WAN*.

At the gateway between *LAN*, *MAN* and *WAN* the speed difference between *WAN* and *LAN* is likely to be large, so that a gateway will be able to transmit packets faster than it receives them. However, the potentially huge number of recipients may exceed the capabilities of the network and must be considered.

If the method used for gathering multicast replies is time distributed, then the returning replies from outside the LAN must be made to conform with the backoff scheme used. Logic would suggest that the best place for this is at the interface between local and remote networks. However, bridges and gateways are often not accessible for change. Therefore, the use of a special WAN agent, which is responsible for forwarding multicasts over WANs may be used as well to gather replies. By substituting the address of the agent for the address of the originator, any replies would be directed to the WAN agent, which can then be forwarded to the appropriate originator, the agent operating the same backoff scheme as local recipients.

One of the features of a hierarchical structure is the isolation afforded to sub-groups. This can be used to advantage when extending the structure to a WAN environment. By using one branch of the tree as a WAN forwarding agent, the characteristics of a WAN can be, to some degree, isolated from the other group members, with the parent of a WAN sub-group appearing to be just a slower host. The WAN agent can then use strict hierarchical forwarding over multiple connections.

The choices in extending multicast over WANs depends on the service provided by the internetworking layer. If only point to point connections are allowed then the multicast is forced to make multiple connections to the destination groups. On the other hand a broadcast capable network may not offer as good a deal as would be supposed, especially if the broadcast is implemented above a point to point network, as in the IP internetwork architecture.

A problem with linking a hierarchical network over a WAN is the fact that the structure described above is only really efficient if all of the level zero members are resident on the same extended LAN. If a group is built arbitrarily then performance can suffer.

The best method for linking a group across a WAN is to set up a WAN *agent* which is responsible for connecting to the WAN agent at other sites and which are the sole members to do so. When a WAN agent receives a message from the WAN it forwards it to the group by acting as the originator of the message. Thus, as the group local to the agent replies or acknowledges, the agent returns the messages across the WAN to the original agent. At the originating site the agent is acting as a level zero host and funnels replies from the WAN and then acts as described above. Therefore a WAN agent is, in effect, connected to the tree in two places, as a level zero host and as a permanent user. Because the agent is a member of the group as well as acting as a user, all of the group operations can be applied to it, although the actions for leaving are different, because it would be difficult to accept a level zero host which resides over a WAN. Therefore, instead of the leaving host being succeeded by one of its children, either a child of one of the other level zero hosts must be elevated to take its place or a level zero host takes over the task of providing WAN access. The new agent must make connections with each of the sites in the group which may involve some extra information being held by the level zero hosts about the connectivity.

If a hierarchical tree is built on a per LAN basis, then the WAN agents act as surrogates for the user on the other side of the WAN. That is any messages that come from the WAN are then retransmitted to the destination group *as if it came from the WAN agent*, so that replies are directed to the WAN agent which then forwards them to the real originator. In addition, acknowledgements are fielded locally, reducing WAN traffic. For such a scheme to operate the WAN agent must either retain knowledge about current incomplete operations or use the packet header to identify the originator directly. The latter would be more secure in the event of an agent failure but requires additional per packet overhead.

If the WAN directly supports multicast then the tree building algorithm could be extended, building unbalanced trees based on topology available and routing algorithms. However, this may be undesirable if originators are resident over the WAN, with all operations having to traverse the network to get to the level zero recipients, which may be all located remotely.

In a transaction oriented protocol a multicast results in each group member replying to the user. The implication of this is that messages from all group members must somehow be transmitted to the user. But in a hierarchical structure a child does not necessarily have a direct path back to the user and will send its reply to its immediate parent. The parent then has the responsibility of forwarding the replies from its children to its parent, as well as sending its own reply to its parent. The parent has the choice of sending each received message individually, or can apply some filtering algorithm to reduce return traffic, such as concatenating messages to form a larger block of messages which is then sent as a single message up to its parent. To enable a child to direct its reply through a parent the final destination address must be present in the packet as well as the, what is in effect a first hop address, parents address.

6.5 Summary

Large scale multicast, especially when transactional operation is required, requires careful structuring of replies in order to prevent implosion. On low speed networks, implosion is less of a problem, as current host technology is well able to cope. However, the trend in networks is for ever more speed while the available processing speed increases more slowly. The effect of packet length on implosion indicates that positive acknowledgements are the major problem when discussing multicast.

A number of possible methods have been described which may be used singly or in conjunction to reduce implosion if it occurs. The use of a hierarchy to reduce the number of replies to any host was described in detail, with the group management operations that are required to maintain the structure.

Chapter 7
Conclusions

A model for implosion was presented which showed that positively acknowledged and transactional multicast could be limited by the buffering capacity of the multicast originator. The model showed how implosion was dependent on the processing delay of packets as well as the transfer rate of computer buses. A specific network architecture was shown to exhibit packet loss when multicasting, this packet loss such that a multicast under the conditions described would never complete. A number of techniques for reducing the problem of implosion, and therefore aiding in protocol scalability, were also described.

7.1 Summary

Motivation.

With distributed systems becoming ever more popular, the advantages of using some form of multicast to provide the data transport service, multicast addressing allowing fault tolerance and location transparency, leads to the view that multicast will become more predominant. This shift towards greater distribution is supported by the increasing transfer rates exhibited by many of the newer network architectures being proposed, with transfer rates in excess of 100 *Mbps* being seen. These two trends lead to a perceived need to address the scalability of multicast, the former because of the potentially large groups that may emerge from this increasing usage of multicast, the latter because the effect of these increasing transfer rates, normally seen as beneficial to communications, decreasing the delay and increasing the data throughput, may be detrimental to multicast.

From the literature survey presented in Chapter 2, one of the observations made was that scalability, described as the maximum group size supported, was treated empirically at best, with the group size being casually stated.

A Multicast Model

The most common model for multicast is described as a one to many type of communication. If the underlying communication is reliable and the communication follows this model then this model is sufficient. However, if the underlying network is not reliable, or the protocol is transactional, then this model does not describe the whole interaction. By including reply traffic, be it acknowledgements for reliability, or data replies, then a possible scalability problem emerges, that of implosion at the multicast originator due to these replies.

Implosion refers to the concentration of replies at a host within a short time period. Implosion is a potential problem if the number of these replies and the time interval within which they arrive combine to cause replies to be discarded due to a lack of buffering in the host. Implosion is a problem for communication in general as evidenced by the use of flow control by protocols. Implosion is a particular problem for multicast for two reasons. The first is due to the synchronizing effect of a multicast, where a message arrives at each group member within a short time, these group members then replying, again within a short time. By implication this is more a problem where the propagation delay between originator and the furthest group member is small, as in a *LAN* or *WAN*. In addition, as the group size becomes larger, it is more likely that the large number of these replies would usurp other traffic, so that a large number of these replies are transmitted back to back on the network, that is with the minimum of inter-packet gaps set by that network. The second problem is due to the replies originating from many group members. This has two aspects. The first concerns acknowledgements, which provide the house keeping functions of a protocol such as flow control, which are not traditionally subject to flow control themselves. The second concerns data replies which may be subject to flow control, but due to the nature of multicast communication, any flow control message is directed to *all* of the group members. Of these the former is of greater concern. If the number of replies is less than causes buffer overrun then there is no problem, if the average rate at which replies arrive is lower than the rate at which the host is able to consume them, then again there is no problem. As described above, multicast is subject to both of these possibilities.

Parameters of Implosion

The influence that a number of parameters have on implosion were investigated. Firstly to show that there was a potential problem, and secondly to show where this problem occurred. In addition, it was hoped that some measure of the problem, and therefore indirectly some measure of multicast scalability, may be devised. Two areas of concern were identified, the first being the role of a network architecture in scalability, the Ethernet type of network exhibiting potential scalability limits. The other area was the host itself, which was modelled as a system of buffers interconnected with data paths of finite transfer rate, the aim being to investigate not only the severity of any implosion but also the location. A simulation of both of these systems was carried out, the results being presented in Chapter 4. The effect that network characteristics have on implosion at a host were of interest given the continual improvement in data transfer rates and other parameters. While such improvements are beneficial for much communication, for multicast they may reduce the scalability offered by a design.

Designs

Having ascertained parameters needed to cause implosion, some means of reducing or mitigating the effect of this implosion was investigated. A number of techniques are described which may be employed in this role. The use of negative acknowledgement protocols for scalable multicast is described. For positively acknowledged and transactional protocols, two basic approaches were investigated. The first concerned increasing the time between replies so that implosion does not result in buffer overrun, the second into distributing the replies among other hosts in order to ensure that the number of replies directed to any host does not cause buffer overrun. These designs are intended to demonstrate how implosion may be overcome, rather than as a blue print for a protocol design, although some detail is expressed in order to demonstrate the operation of these methods.

7.2 Multicast: A General Communication Model

The design of any protocol is driven by the requirements of the application or class of applications that are to use the service provided by the protocol. For multicast protocols the scalability of the design forms one of these application requirements. Scalability has to be considered as an application should not be required to limit its activity because the underlying protocol is unable to operate as required. Scalability is a reliability issue in that reliability mechanisms, such as acknowledgements, are a potential problem, protocols not designed, or at least considered, with scalability as a design issue cannot in turn be regarded as reliable. Whether scalability is required by the application is a separate issue. An application which only requires one reply from some multicast service, any reply satisfying the applications requirements, is unlikely to exhibit the problems investigated.

However, multicast is becoming a communication model applicable to many application classes, such as distributed operating systems, multi-media and fault tolerant computing. By considering multicast as a general communication model that encompasses unicast as well as broadcast, the location independent property of the multicast address becomes a useful mechanism for network transparency, where the location of an application is abstracted from the user. Location independence also implies *migration*, where objects may be migrated from location to location without loss of service. Migration is of interest for storage management, where files are migrated from tape to remote disk to local disk on demand, which implies the use of some form of directory to track the movements of a particular file. This use of multicast would potentially require a large number of multicast addresses, which has been identified as a potential scalability problem, albeit a minor one. Fault tolerance is a use of multicast that employs the grouping of objects to ensure availability. This use of multicast is the more usual interpretation of multicast, and is most often the justification for using a multicast technique over multiple unicasts. Fault tolerance is most often a small scale operation, where a few copies of the object are employed. However, any operations on the object are also most often required to be reliable, and although the scale is small, the potential for implosion and buffer overrun is present. Large scale multicasting applications, such as conferencing, file transfer, file distribution and the like, are more likely candidates for the problems described previously. With increasing distribution and the evolution of multi-media the scalability issues described here become more important.

The literature indicates that there lacks a framework around which multicast functionality may be built. Figure 1.1 shows the issues identified as affecting the design of a multicast protocol. Of these group management is a likely candidate for this framework, and should form the basis for a design rather than something that is employed as an "add-on" to enable the protocol to operate. A number of issues may be considered to be within the domain of group management, among them security and access permission as well as more mundane functions such as the mapping between application and group address. A likely structure for group management is one where a separate *group management* group, using a well known multicast address, is used for the creation and destruction of multicast addresses, so that a unique group address may be guaranteed, as well as performing the mapping between application name and multicast address, acting as a binding service. The *create* and *destroy* group operations are obvious candidates for this group manager. The other operations may also interact with the management group, thus controlling access to the group's services and also providing a known location for address binding, although these may be more easily integrated into the group itself, so that once a group is created, it is the members of the group which manage joining, leaving and failure detection, reducing traffic to the group manager.

The need for order by multicast is an application issue. Placing the ordering paradigm in the protocol rather than the application allows some of the overhead involved in guaranteeing order to be subsumed into the message passing process, as well as allowing the simplification of the application. Of the several ordering paradigms described, the most popular is group order where messages are ordered identically at each group member, which is obviously useful for updating replicated objects. A feature of this order is that the actual order achieved is arbitrary, that is the order is not determined by the user. While desired order is presented as being separate from the others, it may also be employed in conjunction with group order, producing desired group order, where some higher authority imposes an order that is identical at each group member. Order impacts on the scalability of a protocol because of the extra communication and processing overhead imposed.

Figure 7.1 shows a notional multicast protocol with internal layering, which is intended to illustrate how the various design issues described here are may be related within a protocol.

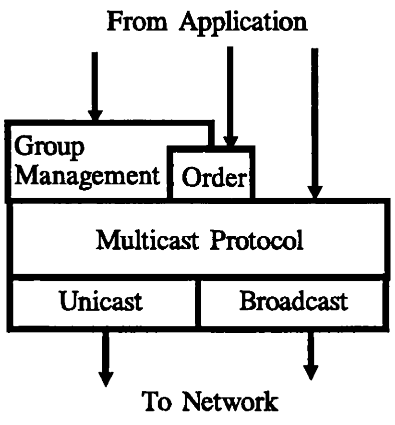


Figure 7.1 A Notional Multicast Protocol Stack

This investigation into the scalability of multicast protocols has indicated, but not proven, that implosion is a potential problem to the scalability of multicast protocols that employ positive acknowledgements or are transaction oriented. One of the aims of this work was to identify parameters for implosion which most affect the degree of implosion exhibited by a particular network/host combination. The network itself may be characterized by two parameters; the transfer rate and the minimum inter-packet gap. A packet is characterized by its length. The overall effect of these parameters is to describe the time it takes for a packet to be received from the network. Each of these parameters was found to be significant to implosion, the most interesting being the effect of packet length on implosion, implosion disappearing for short packets if the time taken to transfer the packet over the bus is shorter than the inter-packet gap time. It was also shown that a length dependent peak occurred in the implosion characteristics of the network buffer module. The bus transfer rate also affected the implosion characteristics of the network buffer, increasing transfer rate decreasing the implosion suffered. The effect of protocol processing on implosion was less evident, although implosion was shown to occur, due partly to the large range chosen as well as the behaviour of the simulation when the bus was saturated, that is whenever the network buffer was also in implosion. The final parameter simulated was the effect of application delay.

The potential for buffer overrun is not solely caused by implosion. The effect of the application, more accurately the interaction between operating system and application, is also of importance. Because the application has to read the packets from the protocol, and that the time between reads may vary and be relatively long, any unread packets are buffered by the protocol, which overflows if the application cannot read packets faster than the protocol receives them. As the network rate increases however, more overruns will be caused by the inability of the host hardware and operating system to process packets, resulting in packets being buffered awaiting protocol processing rather than awaiting the application. As network rates increase further buffer overflow occurs in the interface between network and bus. Of course, if host technology keeps pace with network transfer rates then many of these problems will not occur. From the simulation it was apparent that the main location of implosion is at the protocol level. One of the features of this level is that the amount of buffering can be increased on request to the operating system, thus shifting the goalposts in determining the effect of implosion.

Using a simulation allowed greater access to a number of variables that would not otherwise be available, as well as environments in which to test the model. However, a simulation is an abstraction of the real machine and as such must be treated as such. While the results indicated the behaviour that may be expected of a real host the actual values must be considered with that in mind. In particular the results for the protocol processing assumed a number of features which were optimistic with regard to the delays which may be encountered in a real system. However, the model used indicated the type of behaviour which could be expected, and would probably benefit from a more rigorous examination of this area.

Using simulation to investigate buffer overflow indirectly, by assuming an infinite number of buffers, concentrating on the rate of buffer increase when implosion occurs, allowed a more general view of buffer overflow to be derived. A measure of the severity of implosion was derived by comparing the rate at which buffers were occupied by packets to the generation rate of those packets, resulting in a number, the *implosion index.* The implosion index ranges from 0, indicating no implosion, to 1 indicating that the increase in buffer usage is equal to the rate of packet generation. The index is independent of any buffering, being solely a measure of the difference in the production and consumption of packets at a host. The implosion index relates to positively acknowledged and transactional protocols, where replies to a multicast are always generated by each recipient, a separate index being applied to each buffering system. This index may be used to predict the number of buffers required to receive all of these replies using a simple formulae:

$$\mathit{Number\ of\ Buffers\ Required} = \mathit{Implosion\ Index} \times \mathit{Number\ of\ Replies} + 2 \qquad 7.1$$

In turn, the implosion index may be applied in reverse, allowing the maximum group size which may be supported by any host to be predicted from the number of available buffers at that host. The implosion index may also be applied to the original multicast describing the number of buffers required by the recipient to receive a message acting as a flow control method.

7.3 Multicast Scalability

A number of tentative conclusions may be drawn from this work. The first is that scalability is one of the design issues which must be considered in the design of a protocol. The lack of a formal framework to the design process may be inferred from the protocol designs discussed, many of which ignored the issue of group management, considered here to form the main structural backbone of a well integrated multicast protocol. The issues introduced in chapter 1 are intended to place multicast within a necessary framework.

The host model used to investigate the parameters of implosion represented an attempt to model the structure and operation of a real host. So doing allowed the problem of implosion to be more directly related to the real environment so that the results would be more easily assimilated when considering the design of a protocol. Because of the lack of knowledge about the parameters of implosion, the values employed were deliberately given a wide latitude, which resulted in a relatively sparse spread of results, justified by the need to locate the presence or absence of implosion effects.

The thesis that implosion is a factor in scalability under certain conditions was demonstrated, although the exhibited problems were more apparent with the higher network rates generated for the simulation. However, the idea of an implosion index, which may be employed to predict the potential for buffer overrun, is of interest not just for multicast but in any congestion situation, the use of rate of increase in buffer usage in the simulation being a more useful way of looking at buffer overrun than observing buffer overrun directly. Simulation of the Ethernet showed that there exists a finite limit on the number of recipients of a multicast that may be supported on a single Ethernet, although the value of this limit is sufficiently high not to be a problem in practice.

Several ways of reducing implosion were described. The method which offers the most potential was the loose hierarchical method, which allowed a network to operate at its normal speed while reducing the potential for implosion by reducing the number of replies to any one host. A hierarchical structure is also scalable in the true sense of the word allowing a wide range of group sizes to be supported transparently, the "extra" group members being abstracted by the hierarchy.

Multicast as a general communication model offers a number of advantages over a mixed environment of unicast, multicast and broadcast models. Examples of the potential for multicast can be seen in the use of mobile telephones, with incoming calls being routed to the nearest cell, each cell being informed of its current client list by the 'phones themselves, and by extension to the personal communication concept, allowing each person to possess a unique number throughout their lifetime. Using multicast for these situations would allow caller grouping and conferencing automatically.

7.4 Limitations and Future Investigation

This work has identified a number of design issues which affect scalability. Further work into the implosion problem is required for this to be considered a major problem. Other areas such as the scalability of group management and ordering paradigms also need to be explored in order to show that scalability is a potential problem for multicast systems.

Testing of the host model showed that there were a number of implosion parameters. From these the notion of an implosion index was formulated which could be employed as a measure of the implosion characteristics of a particular host architecture. However, the implosion index is influenced by the network parameters and is therefore not an independent measure of the characteristics of the host. In addition each buffering level was described by a different implosion index. These limit the applicability of the implosion index to a map whre every host/network combination is present, where variations in network and packet processing are used within a model to generate the appropriate implosion index value. This map could then be used to manipulate the characteristics of multicast replies to ensure that buffer overrun does not occur.

One suggestion for further work is to extend the investigation described here to increase the resolution of the system. As this work was intended to explore the boundaries of implosion, wide ranges of values were used to ensure that these boundaries were covered. This necessitated a relatively sparse range for the parameters of implosion, which coupled with the number of parameters resulted in a less detailed view of the problem. A refinement of the simulation model to reduce the number of levels and unify the activities of a host would use a three level model, made up of a producer, the network, a buffer module which exhibits delay, and the consumer which may also have some transfer and delay characteristics. In addition, the simulation used fixed generation rates. As most network architectures are not as deterministic as this, investigating the implosion index with a variable generation rate would also be useful.

An area which would merit further investigation would be the application of the implosion index concept to flow control, where the implosion characteristics of the recipient could be used to control data flow. For example, the originator could tailor its transmission policy, based on the implosion index of the recipient, to ensure that packets are not lost because of buffer overrun. It may also be possible to apply the implosion index to a routing network, where a route is described by its implosion characteristics, possibly on a dynamic basis.

Much of the complexity of multicast, such as group management and order, were subsumed into the data transfer activity by assuming that each of these are higher level entities which used the data transfer service provided to operate. The scalability of group management and ordering protocols is expected to further reduce the overall scalability of a design, as was touched upon earlier. Because of the large number of different variations possible this is an area which requires further investigation.

More generally, the issue of scalability requires more investigation. This dissertation considered that implosion formed the most important limiting factor in the scalability of multicast. As technology advances, the issues described here as being important for scalability may become less important, but the issue of scalability may not.

Extending Multicast to Routing Networks

One of the consequences of considering multicast as a generalized model for communication is the need for the routing of multicast packets across a wide variety of architectures. Modifications to allow such routing in the Internet architecture, bridged *LAN*s and closely coupled networks of micro-processors were described in chapter 3. If multicast is to truly be a generalized communication model, then the efficient routing of multicast packets must be investigated. It is suggested that multicast routing would be a useful area for investigation and by extension, any investigation into multicast routing would also include message passing between different network architectures and the problems that may occur at the network boundary. The scalability of routing protocols for multicast is of obvious importance, with the need to enable messages to be directed to potentially large numbers of recipients, with the attendant possibility of implosion at the routing nodes because of the number of replies traversing the reverse path to the originator. One potential avenue to combat this would be to use different reverse paths back to the originator to reduce traffic through routing nodes.

With the question of routing comes the problems of global address management. The latency of typical store and forward networks would tend to delay the registration of new addresses and the notification of address location changes possibly resulting in inconsistency among group members. As has already been mentioned, the techniques for maintaining consistency may also be susceptible to this latency, indeed in many cases are totally unworkable across inter-connected networks. The creation of a new group has to be co-ordinated across the extended network so that messages are not mis-routed. The dynamic nature of multicast addresses could potentially cause many routing changes to occur, which may result in the routing being unstable during multiple changes.

Multicast routing has also been extended to banyan style switched networks although a number of issues has not been adequately explored with regard to connecting many such switches together. One problem is that there is no longer a simple address which describes the route through all of the connected switches. Because multiple branches can be made, the simple "address is the route" is no longer simple. By using higher level logical addresses which are mapped at each switch to the multicast route through the switch this problem can be overcome. However, the problem now becomes one of tracking this mapping in a dynamic multicast environment where routes are changing and having to be propagated to every connected switch.

Multicast is a powerful technique for many of the issues being investigated. As a generalized model of communication the technique offers a unified view of communication. As a mechanism to group and address programs the technique offers a further level of abstraction over multiple unicast. As a technique to reduce packet traffic by amortizing the cost of a transmission over a number of recipients the method potentially allows a real reduction in traffic for large scale transfers.

Appendix A

Recipient Delay:0.0, Length 2500, Packet Size 1024 Bytes

No.of Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
10	13.0	18.8	10.0	0.0	0	0	17
50	78.3	100.9	67.2	0.0	0	0	96
100	132.0	152.6	120.7	0.0	0	0	254
150	179.7	196.4	163.8	0.1	1	0	366
200	233.4	250.6	213.9	2.1	4	0	497
250	266.2	285.0	247.4	23.0	31	19	849
500	416.9	439.2	395.5	207.0	211	205	2926

Table A.1 Data for Figure 5.2 Reference Curve

Recipient Delay 1.0 msec, Length 2500, Packet Length 1024 Bytes

No. of Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
100	131.7	141.6	124.2	0.0	0	0	266
150	181.1	191.7	174.1	0.1	1	0	457
200	230.9	245.7	218.2	2.0	4	0	639
250	269.3	284.4	253.9	22.0	26	15	831

Table A.2 Data for Figure 5.2 Delay 1.0 msec

Recipient Delay 10.0 msec, Length 2500, Packet Length 1024

Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
200	243.5	257.4	233.4	1.9	5	0	739
250	265.1	281.9	253.0	26.0	28	18	918

Table A.3 Data for Figure 5.4

Recipient Delay 100.0, Length 2500, Packet Length 1024 Bytes

Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
200	275.6	287.1	261.4	1.0	4	0	1007
250	314.7	319.7	302.1	17.8	22	14	1266

Table A.4 Data for Figure 5.4

Recipient Delay 0.0, Length 2500, Packet Length 46 Bytes

Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
200	108.1	119.7	97.1	0	0	0	956
250	138.3	167.3	115.2	0	0	0	1409
300	168.9	198.9	138.3	0	0	0	—
400	307.6	332.9	271.8	45	64	30	—

Table A.5 Data for Figure 5.5

Recipient Delay 0.0, Length 500, Packet Length 1024

Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
200	211.9	224.2	198.9	0.0	1	0	308

Table A.6 Data for Figure 5.6

Recipient Delay 0.0, Length 2500, Packet Length 1024 Bytes, Traffic

Recipients	Completion Time (msec)	Max	Min	Failures	Max	Min	Collisions
200	229.7	240.1	219.0	2.0	0	4	624

Table A.7 Data for Figure 5.7

Implosion at Network Buffer, Transfer Rate 100 *Mbps*

Gap (msecs)	Bus Rate (*MBps*)	Buffer: Usage	Max	Min
1000	10	0.5	1	0
100		1.5	2	1
10		945+/-1		
1		2088+/-1		
0.1		2216+/-1		
1000	25	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	50	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	75	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	100	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1

Table A.8 Data for Figure 5.9

Implosion at Network Buffer, Transfer Rate 1000 *Mbps*

Gap (msecs)	Bus Rate (*MBps*)	Buffer: Usage	Max	Min
1000	10	0.5	1	0
100		1.5	2	1
10		44772+/-3		
1		97348+/-5		
0.1		108776+/-6		
1000	25	0.5	1	0
100		0.5	1	0
10		30123+/-4		
1		45296+/-11		
0.1		94127+/-7		
1000	50	0.5	1	0
100		0.5	1	0
10		5709+/-6		
1		58286+/-10		
0.1		69709+/-10		
1000	75	0.5	1	0
100		0.5	1	0
10		0.5	1	0
1		33873+/-14		
0.1		45295+/-12		
1000	100	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		9464+/-20		
0.1		20884+/-13		

Table A.9 Data for Figure 5.10

Effect of Packet Length, Network Rate 1000 *Mbps*, Gap 1.0 μsec

Data Length (Bytes)	Bus Rate (*MBps*)	Rate of Buffer Increase (Buffers/sec)
64	50	0
	100	0
100	50	14385+/-64
	100	0
128	50	70635+/-54
	100	0
256	50	117980+/-54
	100	0
512	50	93180+/-20
	100	0
1024	50	58286+/-10
	100	9464+/-20
2046	50	32666+/-10
	100	8227+/-14
4096	50	17280+/-10
	100	5076+/-12
8192	50	8891+/-10
	100	2790+/-6

Table A.10 Data for Figure 5.11

Effect of Packet Length on Protocol Processing,
Network Rate 10 *Mbps*, Gap 1.0 μsecs, Bus 50 *MBps*

Data Length (Bytes)	Processing Delay (msec)	Rate of Buffer Increase (Buffers/sec)
32	0.1	5004+/-11
64	0.01	1/0
128	0.1	0
256	0.1	0
512	0.1	0
1024	0.1	0
2048	0.1	0
4096	0.1	0

Table A.11 Data for Figure 5.12

Effect of Protocol Processing, Data Length 1024 Bytes, Network Rate 10 *Mbps*

Bus Rate	Gap	Processing Delay	Buffer
(*MBps*)	(μsecs)	(Buffers/sec)	
10	10000	0.1	0
	100	0.1	0
	10	0.1	—
	1.0	0.1	—
	0.1	0.1	—
25	1000	0.1	0
	100	0.1	0
	10	0.1	713+/-6
		0.01	0
	1.0	0.1	1854+/-5
		0.01	0
	0.1	0.1	1980+/-5
		0.01	0
50	1000	0.1	0
	100	0.1	0
	10	0.1	714+/-5
		0.01	0
	1.0	0.1	1855+/-5
		0.01	0
	0.1	0.1	1979+/-6
		0.01	0
75	1000	0.1	0
	100	0.1	0
	10	0.1	714+/-5
		0.01	0
	1.0	0.1	1854+/-5
		0.01	0
	0.1	0.1	1984+/-6
		0.01	0
100	1000	0.1	0
	100	0.1	0
	10	0.1	709+/-6
		0.01	0
	1	0.1	1855+/-5
	0.01	0	
	0.1	0.1	1981+/-5
		0.01	0

Table A.12 Data for Figures 5.13 & 5.18

Implosion at Protocol, Network Rate 100 & 1000 *Mbps*, Gap 1 μsec.

Bus	Proc	100	Max	Min	1000
10	0.1	9765+/-4			9765+/-1
	0.01	9762+/-2			9762+/-2
	0.001	9763+/-1			9763+/-1
	0.0001	9763+/-2			9763+/-2
25	0.1	1854+/-3			14414+/-4
	0.01	0.5	1	0	24406+/-3
	0.001	0.5	1	0	24403+/-4
	0.0001	0.5	1	0	24405+/-3
50	0.1	1854+/-3			38828+/-5
	0.01	0.5	1	0	48820+/-4
	0.001	0.5	1	0	48820+/-3
	0.0001	0.5	1	0	48820+/-3
75	0.1	1853+/-3			63244+/-5
	0.01	0.5	1	0	73224+/-5
	0.001	0.5	1	0	73234+/-2
	0.0001	0.5	1	0	73230+/-3
100	0.1	1853+/-3			87646+/-5
	0.01	0.5	1	0	97656+/-4
	0.001	0.5	1	0	97645+/-4
	0.0001	0.5	1	0	97645+/-3

Table A.13 Data for Figure 5.14

Effect of Packet Length on Protocol Processing,
Network Rate 100 *Mbps*, Inter-packet gap 10 msecs, Bus 50 *MBps*

Data Length (Bytes)	Processing Delay (msec)	Rate of Buffer Increase (Buffers/sec)
64	0.1	50381+/-6
	0.01	0
	0.001	0
	0.0001	0
128	0.1	36126+/-10
	0.01	0
	0.001	0
	0.0001	
256	0.1	21327+/-7
	0.01	0
	0.001	0
	0.0001	0
512	0.1	9085+/-7
	0.01	0
	0.001	0
	0.0001	0
1024	0.1	714+/-5
	0.01	0
	0.001	0
	0.0001	0
2048	0.1	0
	0.01	0
	0.001	0
	0.0001	0
4096	0.1	0
	0.01	0
	0.001	0
	0.0001	0

Table A.14 Data for Figure 5.15

Effect of Packet Length on Protocol Processing,
Network Rate 100 *Mbps*, Inter-packet gap 10 msecs, Bus 50 *MBps*

Data Length	Processing Delay	Rate of Buffer Increase
(Bytes)	(msec)	(Buffers/sec)
64	0.1	122282+/-14
	0.01	32269+/-21
	0.001	0
	0.0001	0
128	0.1	68865+/-12
	0.01	0
	0.001	0
	0.0001	0
256	0.1	33628+/-10
	0.01	0
	0.001	0
	0.0001	0
512	0.1	13042+/-10
	0.01	0
	0.001	0
	0.0001	0
1024	0.1	1855+/-5
	0.01	0
	0.001	0
	0.0001	0
2048	0.1	0
	0.01	0
	0.001	0
	0.0001	0
4096	0.1	0
	0.01	0
	0.001	0
	0.0001	0

Table A.15 Data for Figure 5.16

Effect of Process Delay, Network Rate 50 *Mbps*, Gap 1 μsec

Processing Delay	Process Delay	No. of Buffer
(msec)	(msec)	(Buffers)
0.01	0.1	2
	1.0	12
	10	118
0.001	0.1	2
	1.0	12
	10	118
0.0001	0.1	2
	1.0	12
	10	118

Table A.16 Data for Figure 5.17

Implosion at Network Buffer, Transfer Rate 10 *Mbps*

Gap (msecs)	Bus Rate (*MBps*)	Buffer: Usage	Max	Min
1000	10	0.5	1	0
100		1.5	2	1
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	25	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	50	0.5	1	
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	75	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1
1000	100	0.5	1	0
100		0.5	1	0
10		1.5	2	1
1		1.5	2	1
0.1		1.5	2	1

Table A.17 Not Used

Effect of Process Delay

Bus (*MBps*)	Delay (msec)	Buffer (Buffers/sec)
25	0.1	1853+/-3
	1	2007+/-10
	10	20744+/-68
50	0.1	1853+/-3
	1	2013+/-10
	10	2159+/-67
75	0.1	1853+/-3
	1	2002+/-10
	10	2168+/-68
100	0.1	1853+/-3
	1	2006+/-6
	10	2086+/-68

Table A.18 Not Used

Effect of Protocol Processing, Data Length 1024 Bytes, Network Rate 10 *Mbps*

Bus Rate (*MBps*)	Gap (μsecs)	Processing Delay (Buffers/sec)	Buffer
10	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	0
	0.1	0.1	0
25	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	0
	0.1	0.1	0
50	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	0
	0.1	0.1	0
75	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	0
0.1	0.1	0	
100	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	0
	0.1	0.1	0

Table A.19 Not Used

Effect of Protocol Processing, Data Length 1024 Bytes, Network Rate 10 *Mbps*

Bus Rate (*MBps*)	Gap (μsecs)	Processing Delay (Buffers/sec)	Buffer
10	1000	0.1	0
	100	0.1	9211+/-3
		0.01	9225+/-2
	10	0.1	—
	1	0.1	—
	0.1	0.1	—
25	1000	0.1	0
	100	0.1	0
	10	0.1	—
	1.0	0.1	—
	0.1	0.1	—
50	1000	0.1	0
	100	0.1	0
	10	0.1	—
	1	0.1	—
	0.1	0.1	—
75	1000	0.1	0
	100	0.1	0
	10	0.1	0
	1.0	0.1	—
	0.1	0.1	—
100	1000	0.1	0
	100	0.1	0
	10	0.1	44536+/-12
		0.01	0
	1.0	0.1	—
		0.01	—
		0.001	97639+/-7
	0.1	0.1	—

Table A.20 Not Used

References

Ag84a.

L. Aguilar, "Datagram Routing for Internet Multicasting," *ACM SIGCOMM 1984 Communications Architectures and Protocols,* 14 (2), pp. 58-63 (6th-8th June 1984).

Al87a.

N. Alon, A. Barak, and U. Manber, "On Disseminating Information Reliably Without Broadcasting," *IEEE,* pp. 74-81 (1987).

Ar92a.

S. Armstrong, A. Freier, and K. Marzullo, "Multicast Transport Protocol," *Request for Comments,* 1301, pp. 1-38 (February 1992).

Ba86a.

E. H. Baalbergen, "Parallel and Distributed Compilations in Loosely Coupled Systems: A Case Study," *Proc. Workshop on Large Grain Parallelism* (1986).

Ba88a.

E. H. Baalbergen, "Design and Implementation of Parallel Make," *Computing Systems,* 1, pp. 135-158 (Spring 1988).

Ba85a.

O. Babaoglu and R. Drummond, "Streets of Byzantium: Network Architectures for Fast Reliable Broadcasts," *IEEE Transactions on Software Engineering,* SE-11 (6), pp. 546-554, IEEE (June 1985).

Baa.

H. E. Bal, M. F. Kaashoek, and A. S. Tanenbaum, "A Distributed Implementation of the Shared Data-object Model," *Proc. First USENIX/SERC Workshop on Experiences with Building Distributed and Multiprocessor systems,* pp. 1-19.

Ba87a.

H. E. Bal, A. S. Tanenbaum, and M. F. Kaashoek, "Orca: A Language for Distributed programming," *IR-140,* Dept. of Mathematics and Computer Science, Vrije Universiteit (December 1987).

Ba85b.

A. Barak and A. Shiloh, "A Distributed Load Balancing Policy for a Multicomputer," *Software - Practice and Experience,* 15 (9), pp. 901-913 (Sept. 1985).

Ba86b.

D. Barbara and H. Garcia-Molina, "The Vulnerability of Vote Assignments," *ACM Transactions on Computer Systems,* 4 (3), pp. 187-213, ACM (Aug. 1986).

Bi89a.

K. Birman and K. Marzullo, "The Role of Order in Distributed Programs," *TR,* 89-1001, Dept. of Computer Science, Cornell University, New York (May 1989).

Bi90a.

K. Birman, A. Schiper, and P. Stephenson, "Fast Causal Multicast," *Cornell University Technical Report,* 90-1105, Dept. of Computer Science, Cornell University (April 1990).

Bi87a.

K. P. Birman and T. A. Joseph, "Reliable Communication in the Presence of Failures," *ACM Transactions on Computer Systems,* 5 (1), pp. 47-76, ACM (Feb. 1987).

Bi87b.

K. P. Birman and T. A. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," *ACM Operating Systems review,* 21 (5), pp. 123-138, ACM (8-11 Nov. 1987). Proc. of 11th ACM Symposium on Operating systems Principles.

Bi89b.

K. P. Birman, R. Cooper, T. A. Joseph, K. P. Kane, and F. Schumk, "ISIS A Distributed Programming Environment" (19 Jun 1989). ISIS System Manual, Version 1.2.

Bi84a.

A. Birrel and B. Nelson, "Implementing Remote Procedure Calls," *ACM Transactions on Computer Systems,* 2 (1), pp. 39-59, ACM (Feb. 1984).

Bo88a.

D. R. Boggs, J. C. Mogul, and C. A. Kent, "Measured Capacity of an Ethernet: Myths and Reality," *Computer Communication Review,* 18 (4), pp. 222-234 (August 1988). Proceedings of the ACM SIGCOMM '88 Workshop.

Bo89a.

D. R. Boggs, J. C. Mogul, and C. A. Kent, "Errata for "Measured Capacity of an Ethernet: Myths and Reality"," *ACM Computer Communications Review,* 19 (2)

(April 1989).

Bo91a.

C. Boozer, "Sun's Upscale SPARC 2," *UnixWorld,* 8 (6), McGraw-Hill (June 1991).

Br89a.

R. Braden, D. Borman, and C. Partridge, "Computing the Internet Checksum," *ACM Computer Communications Review,* 19 (2), pp. 86-94 (Apr 89).

By87a.

G. T. Byrd, R. Nakano, and B. A. Delagi, "A Point to Point Multicast Communications Protocol," *STAN-CS,* 87-1146, Dept. of Computer Science, Stanford University, Stanford (Jan. 1987).

Ca88a.

L. Cabrera, E. Hunter, M. J. Karels, and D. A. Mosher, "User-Process Communication Performance in Networks of Computers," *IEEE Transactions on Software Engineering,* 14 (1), pp. 38-53 (January 1988).

Ch84a.

J. Chang and N. Maxemchuk, "Reliable Broadcast Protocols," *ACM Transactions on Computer Systems,* 2 (1), pp. 251-273, ACM (Aug. 1984).

Ch89a.

S. T. Chanson, G. W. Neufeld, and L. Liang, "A Bibliography of Multicast and Group Communications," *ACM Operating Systems Review,* 23 (4), pp. 20-25, ACM (Oct. 89).

Ch83a.

D. R. Cheriton and W. Zwaenepoel, "The Distributed V Kernel and its Performance for Diskless Workstations," *ACM SIGOPS 1983 Proceedings of the 9th Symposium on Operating Systems Principles,* pp. 129-140 (1983).

Ch83b.

D. R. Cheriton, "Local Networking and Internetworking in the V System," *ACM SIGCOMM Computer Communications Review,* 13 (4), pp. 9-16 (Oct. 1983). 8th Data Communications Symposium.

Ch84b.

D. R. Cheriton, "The V Kernel: A Software Base for Distributed Systems," *IEEE Software Magazine,* pp. 19-42, IEEE (April 1984).

Ch85a.

D. R. Cheriton and W. Zwaenepoel, "Distributed Process Groups in the V Kernel," *ACM Transactions on Computer Systems,* 3 (2), pp. 77-107, ACM (May 1985).

Ch85b.

D. R. Cheriton and S. Deering, "Host Groups: A Multicast Extension for Datagram Networks," *ACM SIGCOMM Computer Communications Review,* 13 (4), pp. 172-178 (Sep. 1985). 9th Data Communications Symposium.

Ch86a.

D. R. Cheriton, "VMTP: A Transport Protocol for the Next generation of Communication Systems," *ACM SIGCOMM 1986 Communications Architectures and Protocols,* 16 (3), pp. 406-415 (Aug. 1986).

Ch87a.

D. R. Cheriton, "UIO: A Uniform I/O interface for Distributed Systems," *ACM Transactions on Computer Systems,* 5 (1), pp. 12-46, ACM (Feb. 1987).

Ch88a.

D. R. Cheriton, "VMTP - Versatile Message Transaction Protocol," *Request For Comment,* 1045 (Feb. 1988).

Ch88b.

D. R. Cheriton, "The V Distributed System," *Communications of the ACM,* 31 (3) (Mar. 1988).

Ch89b.

D. R. Cheriton and C. L. Williamson, "VMTP as the transport Layer for High-Performance Distributed Systems," *IEEE Communications Magazine,* pp. 37-44 (Jun. 1989).

Cl82a.

D. Clark, "Window and Acknowledgement Stategy in the TCP," *Request For Comments,* 813 (July 1982).

Cl87a.

D. Clark, M. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol," *Request For Comment,* 998 (Mar. 1987).

Cl87b.

D. D. Clark, M. L. Lambert, and L. Zhang, "NETBLT: A High Throughput

Transport Protocol," *Computer Communication Review,* 17 (5), pp. 353-359 (August 1987).

Cl89a.

D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine,* pp. 23-29 (Jun. 1989).

Co84a.

C. S. Cooper, "Managed File Distribution on the UNIVERSE Network," *ACM SIGCOMM 1984 Communications Architectures and Protocols,* 14 (2), pp. 10-17, ACM (6th-8th June 1984).

Co91a.

J. R. Corbin, "The Art of Distributed Applications," SUN Technical Reference Library, Springer-Verlag (1991).

Cr88a.

J. Crowcroft and K. Paliwoda, "A Multicast Transport Protocol," *ACM SIGCOMM 1988 Communication Architectures and Protocols,* 18 (4), pp. 247-256 (Aug. 1988).

Cr89a.

J. Crowcroft, "Load Sharing and Adaptive Systems," *Internal Note,* 2357, Dept. of Computer Science, University College London (1989).

Cr89b.

J. Crowcroft, "An X Conferencing Program," *Internal Note,* 2358, Dept. of Computer Science, University College London (1989).

Da78a.

Y. K. Dalal and R. M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets," *Communications of the ACM,* 21 (12), pp. 1040-1048 (December 1978).

Da89a.

P. B. Danzig, "Finite Buffers and Fast Multicast," *ACM SIGMETRICS Performance Evaluation Review,* 17 (1), pp. 108-117, ACM (23-26 May 1989).

De85a.

S. Deering and D. R. Cheriton, "Host Groups: A Multicast Extension to the Internet Protocol," *Request For Comment,* 966 (Dec. 1985).

De86a.

S. Deering, "Host Extensions for IP Multicasting," *Request For Comment,* 988

(Jul. 1986).

De88a.

S. Deering, "Host Extension for Multicasting," *Request For Comment,* 1054 (May 1988).

De89a.

S. Deering, "Host Extensions for IP Multicasting," *Request For Comments,* 1112, pp. 1-17 (Aug. 1989).

De91a.

S. Deering, "ICMP Router Discovery Messages," *Request for Comments,* 1256, pp. 1-19 (September 1991).

De90a.

S. E. Deering and D. R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Trans. on Computer Systems,* 8 (2), pp. 85-110, ACM (May 1990).

De89b.

G. S. Delp and D. J. Farber, "Memnet - A Different Approach to a Network," *comp.os.research,* 6934@saturn.ucsc.edu (April 1989).

De91b.

A. DeSimone, "Generating Burstiness in Networks: A Simulation Study of Correlation Effects in Networks of Queues," *ACM Computer Communications Review,* 21 (1), pp. 24-31 (January 1991).

Du90a.

S. Dutt and J. P. Hayes, "On Designing and Reconfiguring k-Fault-Tolerant Tree Architectures," *IEEE Trans. on Computers,* 39 (4), pp. 490-503 (April 1990).

Er87a.

A. Erramilli and R. Singh, "A Reliable and Efficient Multicast Protocol for Broadband, Broadcast Networks," *ACM SIGCOMM 1987 Computer Communications Review,* Vol. 17 (5), pp. 343-352 (11th-13th Aug. 1987). Frontiers of Computer Communications Technology Special Issue.

Fa89a.

D. J. Farber, "Some Thoughts on the Impact of Ultra-High-Speed Networking on Processor Interfaces," *comp.os.research,* 6935@saturn.ucsc.edu (April 1989).

Fe90a.

D. Ferrari, "Client Requirements for Real-Time Communication Services," *Request for Comments,* 1193, pp. 1-24 (November 1990).

Fi91a.

S. Fisher, "Crescendo Speeds Up Data Transmission Without Fibre," *Byte,* p. 24, McGraw-Hill (October 1991).

Fr85a.

A. Frank, L. Whittie, and A. Bernstein, "Multicast Communication on Network Computers," *IEEE Software Magazine,* pp. 49-61, IEEE (May 1985).

Ga86a.

R. Gaglianello and H. Katseff, "Communication in MEGLOS," *Software - Practice and Experience,* 16 (10), pp. 945-963 (Oct. 1986).

Ga89a.

J. Gait, "A Kernel for High Performance Multicast Communication," *IEEE Transactions on Computers,* COM-38 (2), pp. 218-226 (Feb. 1989).

Ga88a.

H. Garcia-Molina and A. Spauster, "Message Ordering in a Multicast Environment," *CS-TR,* 161-88, Dept. of Computer Science, Princeton University (June 1988).

Ga88b.

H. Garcia-Molina and A. Spauster, "Ordered and Reliable Multicast Communication," *CS-TR,* 184-88, Dept. of Computer Science, Princeton University (Oct. 1988).

Gi81a.

D. Gifford, "Weighted Voting for Replicated Data," *ACM SIGOPS 1981 Proceedings of the 7th Symposium on Operating Systems Principles,* pp. 150-162 (1981).

Go84a.

I. Gopal and J. Jaffe, "Point to Multipoint Communication Over Broadcast Links," *IEEE Transactions on Communications,* COM-32 (9), pp. 1034-1044, IEEE (Sept. 1984).

Gr78a.

J. Gray, "Notes on Database Operating Systems" in *Operating Systems: An*

Advanced Course, ed. G.Goos and J. Hartmannis, 60, Springer-Verlag (1978).

He88a.

C. Hedrick, "Routing Information Protocol," *Request For Comment,* 1058 (June 1988).

He86a.

M. Herlihy, "A Quorum-Consensus Replication Method for Abstract Data Types," *ACM Transactions on Computer Systems,* 4 (1), pp. 32-53, ACM (Feb. 1986).

Ho87a.

J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West, "Scale and performance in a Distributed File System," *ACM Operating Systems review,* 21 (5), pp. 1-2, ACM (8-11 Nov. 1987). Proc. of 11th ACM Symposium on Operating systems Principles.

Hu88a.

L. Hughes, "A Multicast Interface for UNIX 4.3," *Software - Practice and Experience,* 18 (1), pp. 15-27 (Jan. 1988).

Hu89a.

L. Hughes, "Multicast Response Handling Taxonomy," *Computer Communications,* 12 (1), pp. 39-46 (Feb. 1989).

Ja90a.

V. Jacobsen, R. Braden, and L. Zhang, "TCP Extensions for High-Speed Paths," *Request for Comments,* 1185 (October 1990).

Ja89a.

R. Jain, "A Comparison of Hashing Schemes for Address Lookup in Computer Networks," *DEC-TR-593,* pp. 1-17 (Feb. 1989).

Ja89b.

R. Jain, "Characteristics of Destination Address Locality in Computer Networks: A Comparison of Caching Schemes," *DEC-TR-592* (Feb. 1989).

Jo89a.

S. Johnsson and C-T Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Transactions on Computers,* 38 (9), pp. 1249-1268, IEEE (Sept. 1989).

Jo91a.

M. G. W. Jones, S-A Sorensen, and S. R. Wilbur, "Protocol Design for Large Scale Multicasting: The Message Distribution Protocol," *Computer Communications,* 14 (5), pp. 287-297 (June 1991).

Jo86a.

T. A. Joseph and K. P. Birman, "Low Cost Management of Replicated Data in Fault Tolerant Distributed Systems," *ACM Transactions on Computer Systems,* 4 (1), pp. 54-70, ACM (Feb. 1986).

Jo88a.

T. A. Joseph and K. P. Birman, "Reliable Broadcast Protocols," *Technical Report,* 88-918, Dept. Computer Science, Stanford University (Jun. 1988).

Ka89a.

M. Frans Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal, "An Efficient Reliable Broadcast Protocol," *ACM Operating Systems Review,* 23 (4), pp. 5-19, ACM (Oct. 89).

Ka91a.

D. D. Kandler and K. G. Shin, "Reliable Broadcast Algorithms for HARTS," *ACM Trans. on Computer Systems,* 9 (4), pp. 374-398 (Nov. 1991).

Ku91a.

A. Kumar, "Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data," *IEEE Trans. on Computers,* 40 (9), pp. 1005-1015 (September 1991).

La82a.

L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems,* 4 (3), pp. 382-401, ACM (Jul. 82).

La78a.

L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. of ACM,* 21 (7), pp. 558-565, ACM (July 1978).

Le85a.

K. Lebowitz and D. Mankins, "Multi-Network Broadcasting within the Internet," *Request For Comment,* 947 (June 1985).

Le89a.

S. Leffler, M. McKusick, M. Karels, and J. Quarterman, "The Design and Implementation of the 4.3BSD Unix Operating System," Addison-Wesley (1989).

Li89a.

Y-K. M. Lin, D. R. Spears, and M. Yin, "Fiber-Based Local Access Network Architectures," *IEEE Communications Magazine,* 27 (10), pp. 64-73, IEEE (Oct. 1989).

Lo88a.

D. D. E. Long, J. L. Carroll, and K. Stewart, "The Reliability of Regeneration-Based Replica Control Protocols," *UCSC-CRL,* 88-18, Computer Research Laboratory, University of California (Oct 1988).

Lo88b.

D. D. E. Long and J-F. Paris, "Regeneration Protocols for Replicated Objects," *UCSC-CRL,* 88-23, Computer Research Laboratory, University of Califonia (Oct 1988).

Lo88c.

D. D. E. Long, J-F. Paris, and J. L. Carroll, "Reliability of Replicated Data Objects," *UCSC-CRL,* 88-34, Computer Research Laboratory, University of California (Dec. 1988).

Lo90a.

D. E. Long and J-F. Paris, "Voting with Regenerable Volatile Witnesses," *UCSC Technical Report,* CRL-90-09, UCSC, Santa Cruz (July 1990).

Ma85a.

K. Marzullo and S. Owick, "Maintaining Time in a Distributed System," *ACM Operating Systems Review,* 19 (3), pp. 44-54, ACM (Jul. 1985).

Ma88a.

K. Marzullo and F. Schmuk, "Supplying High Availability with a Standard Network File System," *Proc. 8th International Conference on Distributed Computing Systems,* pp. 447-453 (13-17 Jun. 1988).

Ma90a.

K. Marzullo, M. Wood, R. Cooper, and K. Birman, "Tools for Distributed Application Management," *Cornell University Technical Report,* 90-1136, Dept.

of Computer Science, Cornell University (July 1990).

Mc90a.

P. K. McKinley and J. W. S. Liu, "Multicast Tree Construction in Bus-Based Networks," *Communications of the ACM,* 33 (1), pp. 29-42 (Jan. 1990).

Me90a.

P. M. Melliar-Smith, L. E. Moser, and V. Agrawala, "Broadcast Protocols for Distributed Systems," *IEEE Trans. on Parallel and Distributed Systems,* 1 (1), pp. 17-25, IEEE (January 1990).

Me76a.

R. Metcalfe and D. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Communications of the ACM,* 19 (7), pp. 395-404 (Jul. 1976).

Mi89a.

S. E. Minzer, "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine,* 27 (9), pp. 17-24, IEEE (Sept. 1989).

Mo83a.

P. Mockapetris, "Analysis of Reliable Multicast Algorithms for Local Networks," *ACM SIGCOMM Computer Communications Review,* 13 (4), pp. 150-157 (Oct. 1983). 8th Data Communications Symposium.

Mo84a.

J. Mogul, "Broadcasting Internet Datagrams," *Request For Comments,* 919 (Oct. 1984).

Mo84b.

J. Mogul, "Broadcasting Internet Datagrams in the Presence of Subnets," *Request For Comments,* 922 (Oct. 1984).

Mo87a.

S. Mohan, J. Qian, and N. Rao, "Efficient Point to Point and Point to Multipoint Selective Repeat ARQ Scheme with Multiple Retransmission: A Throughput Analysis," *ACM SIGCOMM 1987 Computer Communications Review,* Vol. 17 (5), pp. 49-57 (11th-13th Aug. 1987). Frontiers of Computer Communications Technology Special Issue.

Mo86a.

J. H. Morris, M. Satyanarayanan, M. H. Conner, D. S. Rosenthal, and F. D.

Smith, "Andrew: A Distributed Personal Computing Environment," *Communications of the ACM,* 29 (3) (March 1986).

Mu88a.

S. J. Mullender, "Distributed Operating Systems: State-of-the-Art and Future Directions," *Proc. of the EUTECO 88 Conference,* pp. 57-66 (1988).

Mu90a.

S. J. Mullender, G. Rossum, A. S. Tanenbaum, R. van Renesse, and H. van Staveren, "Amoeba-A Distributed Operating System for the 1990s," *IEEE Computer Magazine* (May 1990).

Na88a.

S. Navaratnum, S. Chanson, and G. Neufeld, "Reliable Group Communication in Distributed Systems," *Proc. 8th International Conference on Distributed Computing Systems,* pp. 439-446 (13-17 Jun. 1988).

Ne88a.

R. M. Newman, Z. L. Budrikis, and J. L. Hullett, "The QPSX Man," *IEEE Communications Magazine,* 26 (4), pp. 20-28, IEEE (Apr. 1988).

Nga.

L. H. Ngoh and T. P. Hopkins, "Multicast Support for Multimedia Group Communications," *To be published,* Dept. of Computer Science, Manchester University.

Pa88a.

K. Paliwoda, "Transactions Involving Multicast," *Computer Communications,* 11 (6), pp. 313-318 (Dec. 1988).

Pa88b.

G. Pavlou, "Distributed database Study: Design and Implementation Specification," *Internal Note,* 2275, Dept. of Computer Science, University College London (June 1988).

Pl82a.

D. Plummer, "An Ethernet Address Resolution Protocol," *Request for Comments,* 825 (November 1982).

Po80a.

J. Postel, "User Datagram Protocol," *Request For Comments,* 768, pp. 1-3 (Aug. 1980).

Po81a.

J. Postel, "Internet Protocol," *Request for Comments,* 791 (September 1981).

Po81b.

J. Postel, "Transmission Control Protocol," *Request For Comments,* 793, pp. 1-85 (Sept. 1981).

Po83a.

M. Powell and D. Presotto, "Publishing: A Reliable Broadcast Communication Mechanism," *ACM SIGOPS 1983 Proceedings of the 9th Symposium on Operating Systems Principles,* pp. 100-109 (1983).

Ra86a.

S. Ramchandra and S. Lim, "A Selective-Repeat ARQ Scheme for Point to Multipoint Communication and its Throughput Analysis," *ACM SIGCOMM 1986 Communications Architectures and Protocols,* 16 (3), pp. 292-301 (Aug. 1986).

Ro91a.

J. Robinson, E. Rubinov, C. Toulon, B. Prasada, S. Sabri, N. Goldberg, and G. Vanderweidt, "A Multimedia Interactive Conferencing Application for Personal Workstations," *IEEE Trans. on Communications,* 39 (11), pp. 1698-1708 (Nov. 1991).

Ro86a.

F. E. Ross, "FDDI - A Tutorial," *IEEE Communication Magazine,* 24 (5), pp. 10-17 (May 1986).

Ro90a.

F. E. Ross, J. R. Hamstra, and R. L. Fink, "FDDI - A LAN among MANs," *ACM Computer Communications Review,* 20 (3), pp. 16-31 (July 1990).

Sa90a.

M. Satyanarayanan and E. H. Siegel, "Parallel Communication in a Large Distributed Environment," *IEEE Trans. on Computers,* 39 (3), pp. 328-348, IEEE (March 1990).

Sa90b.

M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers,* 39 (4), pp. 447-459 (April

1990).

Sc84a.

F. Schneider, D. Gries, and R. Schlichting, "Fault Tolerant Broadcasts," *Science of Computer Programming*, 4, pp. 1-15 (1984).

So91a.

S-A Sorensen and M. G. W. Jones, "The CLOWN Network Simulator," *Proceedings of the 7th Performance Engineering Conference*, pp. 123-130, Springer-Verlang (1991).

St90a.

W. R. Stevens, "UNIX Network Programming," Software Series, Prentice Hall (1990).

Su86a.

S. Y. Suh, S. W. Granlund, and S. S. Hegde, "Fiber-Optic Local Area Network Topology," *IEEE Communications Magazine*, 24 (8), pp. 26-32, IEEE (Aug. 1986).

T89a.

AT & T, "UNIX System V Release 3.2 - Streams Programmers Guide," Prentice-Hall (1989).

Ta86a.

A. Tanenbaum and R. Van Renesse, "Distributed Operating Systems," *Computing Surveys*, 17 (4), pp. 419-470 (Dec. 1986).

Ta89a.

A. S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp, S. J. Mullender, A. J. Jansen, and G. van Rossum, "Experiences with the Amoeba Distributed Operating System," Report IR-194, Dept. of Mathematics and Computer Science, Vrije Universiteit (July 1989).

Va91a.

G. Vanderweidt, J. Robinson, C. Toulon, J. Mastronardi, E. Rubinov, and B. Prasada, "A Multipoint Communication Service for Interactive Applications," *IEEE Trans. on Communications*, 39 (12), pp. 1875-1885 (Dec. 1991).

Ve89a.

P. Verissimo, L. Rodrigues, and M. Baptista, "AMp: A Highly Parallel Atomic Multicast Protocol," *ACM Computer Communications Review*, 19 (5), pp. 83-93

(Oct. 89).

Wa88a.

D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," *Request For Comment,* 1075 (Nov 1988).

Wa80a.

D. W. Wall, "Mechanisms for Broadcast and Selective Broadcast," *Ph. D. Thesis,* Stanford University (June 1980).