Fast Group Communication for Standard Workstations

Werner Vogels
INESC[†]
Lisboa, Portugal
werner@inesc.pt

Luis Rodrigues   Paulo Veríssimo
Technical University of Lisboa
INESC, Lisboa, Portugal
{ ler | paulov }@inesc.pt

Abstract

This paper presents a Group Communication Service suitable for standard workstations. The communication service is designed to take advantage of the technology offered by modern standard Local Area Networks and offers a very versatile multi-primitive interface to its users. The authors focus on the design and implementation of the communication service, and of the software modules necessary to exploit specific network and operating system properties. Additionally performance results are given and evaluated in the context of comparable systems.

1. Introduction

Increasing use of distributed systems, with the corresponding decentralization of activities, stimulates the need for structuring those activities around groups of participants, for reasons of consistency, user-friendliness, performance and dependability. The concept appears intuitively in all flavors of distributed actions: when participants cooperate in an activity (e.g. management of a partioned database, shared document processing or distributed process control), compete for a given activity (e.g. distributed use of a resource), or execute a replicated activity for performance or fault-tolerance reasons (e.g. replicated database server, replicated actuator).

The group paradigm is widely accepted as being an excellent method of structuring these distributed activities. From the pioneering projects

[†] Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 – 6° – 1000 Lisboa – Portugal, Tel.+351-1-3100281. This work has been supported in part by JNICT, through Programa Ciência.

in the past [Bir91a, Coo85a, Che85a], a large number of research projects in areas related with group structuring and reliable group communication have emerged [Bir91b, Cri90a, Gar89a, Her89a, Pet89a, Pow91a]. The Distributed Systems and Industrial Automation group at INESC has contributed to the evolution of the group paradigm by focusing on the development of highly responsive group communication and management protocols [Ver92a].

To support the development of systems and applications that rely on distributed paradigms, we have developed a Group Communication Service. Originally designed and developed as part of the Delta-4† ESPRIT project [Pow91a], an effort has been undertaken to make the same service available for standard workstations. The results of this effort have yielded a group communication module suitable for integration in UNIX kernels. Prototype implementations have been made for the SunOS 4.1.1 and the Mach 2.5 kernels.

In this paper we discuss the global design and implementation of the Group Communication Service and related modules. The next section describes our approach to group communication in general, followed by a section on the actual design of the service. The different modules that found the basis of the service are each described in separate sections after the section on design. Section 10 will deal with the formal specification and verification of the protocols. In Sections 11 and 12 we present the performance of our protocols and evaluate these results by comparing them to other group communication systems.

2. The Group Communication Service

The need for support of group activity is based on the assumption, shown correct by a number of real examples, that in a distributed architecture processes frequently get together to achieve a common goal. The set of such processes can be called a *group*. A communication service can be said to support groups when it provides services that facilitate the design and the execution of distributed software running on such a group of distributed processes in cooperation, competition or replication [Ver92a].

The Group Communication Service described in this paper is based on three essential services [Rod92a]:

- The first services required in a group communication service are, naturally, the *group membership* services. Powerful support for groups is given to allow the dynamic creation – and reconfiguration – of process groups. During the lifetime of a group, processes may join or leave the group and the communications service provides primitives to perform these operations. The failure of a group member is also detected and an indication of the event is provided to the remaining members.
- The second goal of the group communication service is to provide efficient and versatile support for exchange of information between group members. To start with, a *multicast* communication service avoids the need to explicitly perform point-to-point transfers to execute a multicast operation. The service accepts a list of addresses, what we call a *selective address*, as a valid destination address for a multicast message and – transparently –

† Delta-4, ended in December 1991, was a CEC Esprit II consortium, formed by Ferranti-CSL (GB), Bull (F), Credit Agricole (F), IEI (I), IITB (D), INESC (P), LAAS (F), LGI (F), MARI (GB), NCSR (GB), Renault (F), SEMA (F), Un. of Newcastle (GB), designing an open, dependable, distributed architecture.

	Consistent Group View
Px1	Each change to group membership is indicated by a message obeying total order, to all correct group participants within a known and bounded time T_g.
	Addressing
Px2	**Selective addressing:** The recipients of any message are identified by a pair (g, sl), where g is a group identification and *sl* is a *selective address* (a list of physical addresses).
Px3	**Logical addressing:** For each group g there is a mapping between g and an address A_g, such that A_g allows all correct members of g to be addressed without the knowledge by the sender of their number or physical identification.
	Validity
Px4	**Non-triviality:** Any message delivered, was sent by a correct participant.
Px5	**Accessibility:** Any message delivered, was delivered to a participant correct and accessible for that message.
Px6	**Delivery:** Any message is delivered, unless the sender fails, or some participant(s) is(are) inaccessible.
	Synchronism
Px7	The time between any service invocation and the (eventual) subsequent indication at any recipient (T_e), as well as the time between any two such (eventual) indications (T_i), are: – *Loose synchronism:* ΔT_e and ΔT_i may be not negligible, in relation to max T_e. – *Tight synchronism:* ΔT_e and ΔT_i are negligible, in relation to max T_e
	Agreement
Px8	**Unanimity:** Any message delivered to a participant, is delivered to all correct addressed participants.
Px9	**At-least-N:** Any message delivered to a recipient, is delivered to at least N correct recipients.
Px9.1	**At-least-To:** Given a subset P_{addr} of the recipients, any message delivered to a recipient, is delivered to all correct recipients in P_{addr}.
Px10	**Best-effort-N:** Any message delivered to a recipient, is delivered to at least N correct recipients, in absence of sender failure.
Px10.1	**Best-effort-To:** Given a subset P_{addr} of the recipients, any message delivered to a recipient, is delivered to all correct recipients in P_{addr}, in absence of sender failure.
	Order
Px11	**Total order:** Any two messages delivered to any correct recipients, are delivered in the same order to those recipients.
Px12	**Causal order:** Any two messages, delivered to any correct participants of any group, are delivered in their "precedes" order.
Px13	**FIFO order:** If any two messages from the same participant, are delivered to any correct recipient, they are delivered in the order they were sent.

Table 1: *Group communication properties*

delivers the message to the intended recipients. Additionally, a *logical address* can be associated with a multicast group, allowing all group members to be addressed through a *logical name*. This frees the programmer from having to deal explicitly with selective address lists. Note that a logical name can be seen as a pre-defined address list, containing the addresses of all group members, and being constantly updated upon every group change.

- The third goal of the group communication service is to provide an execution environment that applies algorithms to ensure a given set of desirable properties.† These properties are summarized in Table 1. Validity and synchronism properties (Px4, Px5, Px6 and Px7) are desirable in most communication systems. They usually state that the user can trust the system in the sense that messages are not corrupted, arbitrarily lost or spontaneously generated. Synchronism properties assure that the service is provided within known time bounds. Timely behavior of the protocol is of major relevance in real-time systems. Agreement properties describe when, and to whom, a multicast message must be delivered. The strongest property in this set is *unanimity* (Px8). Unanimity states that a message, if delivered to a correct participant, will be delivered to all other correct participants despite the occurrence of faults. This may be stronger than usually required. For instance, queries to replicated servers need only reach one of the replicas, since all responses would be the same. Quorum-based protocols are another example where unanimity is not required. This raised the need to provide different agreement properties (Px9 and Px10). Finally, order properties specify which ordering disciplines the protocol should impose on the messages exchanged between group members. The stronger property, *total order* (Px11) assures that the messages are delivered in the same order to different participants. Causal (Px12) and FIFO (Px13) are different, less costly, ordering disciplines that can provide better performance for those applications not requiring total order.

Clearly, all these different requirements cannot be provided in an efficient manner by a single communication primitive. That is why the Group Communication Service provides several qualities of service [Rod91a]:

- **Best-Effort**. Acknowledged datagrams with retries to reach a certain quorum. Quorum can be set by either a number of members or a subset of addressees.
- **Reliable**. Acknowledged datagrams with retries and Quorum specification like in Best-effort but with guarantee of delivering even if the sender fails.
- **Causal**. Reliable quality of service respecting the “happened before” order.
- **Atomic**. Datagrams delivered to all members (including the sender) or none with total order within the group.
- **Tight**. Total order datagrams within a group with queue re-ordering for priority handling of messages and approximate same time delivery.
- **Delta**. Support for total order of messages based on global time (achieved by synchronized virtual clocks).

3. Design

The design of the Group Communication Service was driven by a number of goals:

- Exploitation of technology offered by the network infrastructure.
- Offer a versatile set of primitives that can satisfy all application requirements regarding group communication.

† For a more detailed study the reader is referred to [Ver89a].

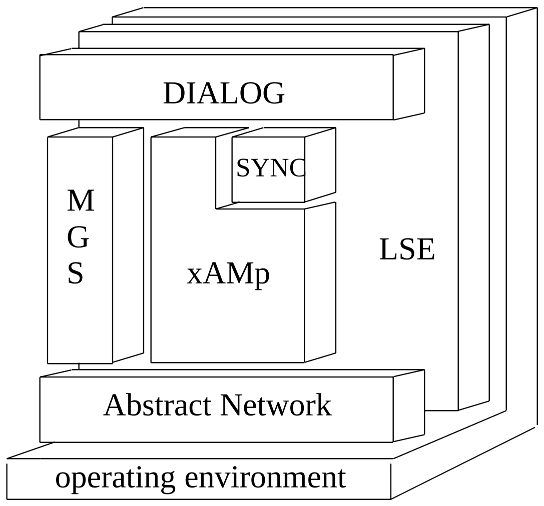

Figure 1: *Modules in the Group Communication Service*

- Highly responsive behavior of all primitives.
- Entry points at all layers are accessible by the user of the service.

Another goal was to design the Group Communication System as a highly portable software system suitable for integration in several different operating environments. User level programming environments are getting more and more standardized, and it is becoming easier to develop software that is portable at this level. At the operating system level the situation is the contrary, manufactures are moving away from the original base (often BSD UNIX) making it more difficult to built portable kernel modules. We have gone through considerable effort to design our Group Communication Service in such a way that the core part of the system is highly portable, and is surrounded by a number of well defined modules that implement the environment dependencies.

The following modules are part of the Group Communication Service (see also Figure 1):

- **Local Support Environment** – Offers an environment independent interface to system specific functions, such as memory allocation, timers, buffer management and event handling [Fon90a].
- **Abstract Network** – This modules implements all network properties common to networks that need to support group communication [Ruf91a]. It handles address management, the sending of messages, filtering of incoming messages, and supplies support for algorithms that are based on properties like *bounded execution time*.
- ***x*AMp** – The core protocol kernel, implementing the Qualities of Service described in Section 2 [Rod92a].
- **MGS** – The Multicast Group of Stations protocol. This is a low level processor group membership protocol designed to support membership and addressing techniques [Rod92b].
- **SYNC** The Clock Synchronization Service. Algorithms are implemented to achieve synchronized virtual clocks, creating a global time base [Rod91b].
- **Dialog** This is the interface module to make the system work with each of the three standard UNIX communication interfaces *socket*, *streams* and *device driver*.

Each of these modules is described in detail in the following sections. In nearly all sections we provide some implementation details. These details are related to the SunOS and MACH 2.5 ports. Ports to other environments have also been made but are out of the scope of this paper as the are not considered as "standard workstations".

4. Local Support Environment

The Group Communication Service is designed to be environment independent, resulting in a highly portable protocol core that has no dependencies to a particular operating environment and a well described interface that covers all possible environment specific interactions. Minimal porting efforts are needed to bring the service to another environment.

To be able to shield the protocols from all environment dependencies a *Local Support Environment* (LSE) has been developed [Fon90a]. This LSE is not only a product of theoretical design, but it reflects our experiences with porting the Group Communication Service to different platforms. Especially in the areas of timer and buffer management the design of the LSE modules have undergone substantial changes through the years, to arrive at a point where they have become generic packages, usable by designers of any protocol, offering more functionality then the underlying operating system provides.

In addition to the modules that implement the interface to the environment dependent system parts, a number of generic data structure handling routines have been integrated into the LSE, adding easy to use *pools*, *lists*, *etc* to the protocol development environment. There is an overhead in making these data structure handling routines generic, but during the design phase it shortens the prototyping path. If during profiling it turns out that the generic manipulation introduces a substantial performance penalty, dedicated implementations of the data structure handling routines are built.

When porting to a different operating system environment, the dependent parts of the LSE need to be re-implemented to match the new environment. The environment dependent modules include:

4.1. Buffer Management

How to construct and manipulate messages is of extreme importance when designing high-performance protocols [Bir84a, Che88a, Hut89a, Dru92a, Sch89a, Ber89a]. Former research pointed out that operations on message buffers are often bottlenecks in the performance of network software. Especially the copy operations are to be avoided.

Although an effort has been made to design the LSE buffer management as efficient as possible, the *avoid to copy* rule dominates the design, resulting in that the operating system specific buffer management scheme is left intact as much as possible.

In the UNIX kernel and the MACH macrokernel versions this resulted in using the LSE buffer management as a frontend to operations on *mbuf's*. The only need for copying is from user to kernel space and from kernel to device space. These are, given the current structure of UNIX, the minimum number of operations that you have to apply. The new mbuf scheme in SunOS 4.1.1, which makes it possible to add private manipulation functions to arbitrary sized mbuf's, looks promising for designing dedicated buffer management. But for the time being it is inadequate because the network devices will still use the old scheme for assigning frames to buffers. To convert to the new style mbuf's an extra copy operation is needed.

The LSE buffer management is implemented as a regular mbuf chain in which the first mbuf has data size zero (0) and contains only administrative data for buffer manipulation. When passing the mbuf chain to standard kernel routines, these routines will discard this first mbuf.

The user of the buffer management is presented a contiguous buffer in which read and write operations can be done at any desired location and headers and tails can be added and removed without caring about the mbuf representation.

4.2. Memory Management

This module presents an interface to allocating and releasing pieces of memory. The routines in this module are merely function calls to the system routines that perform memory allocation. No direct translation can be made because often the *malloc* and *free* calls have different semantics when used in different environments. The SunOS kernel version of *free* for example expects the number of released bytes to be given as an parameter, while most higher level versions of this routine can be satisfied with just an address of the memory block. Some clever tricks have to be used if one wants to keep the interface as efficient as possible. In the UNIX kernel the memory is taken of the kernel heap, which turns out to be out an expensive operation. This justified the design of a local memory management package that would overcome all these difficulties, but adding this type of complexity does not outweigh the advantage. When designing the protocols care has been taken only to use dynamic allocation in startup phases and when there is not critical impact on protocol performance.

4.3. Timer Management

A module with standard timer operations is based on operations on a delta list of timers using the kernel *timeout* function to fire a timer interrupt function.

Timers can be created, destroyed, started and stopped. Two types of timers are available:

- *A-synchronous* timers which execute a registered function at the moment they expire.
- *Synchronous* timers which will send a timeout message to a message queue once they expire.

Using synchronous timers can enhance the simplicity of the protocol code as there is no need for complex interrupt handling of timer triggered routines. Concurrency is locked out of the design of the protocol state machines to simplify the state transition mechanisms.

4.4. Debug and Logging Management

This module offers convenient routines for printing warnings, errors and debug statements. It also provides interface for time measurement to enable intra-kernel performance management.

When operating normally or with a small number of debug messages the system makes use of the syslog facility or writes directly to the console device. When the number of debug messages is expected to become high, the messages can be send to a special *xamp-debug* device that handles the messages very efficiently. Extreme care has to be taken when writing verbose debug messages to the console of a SUN workstation, as this console is so slow that it can effectively block the execution of protocols when printing for longer periods, making the protocol deaf and dumb for unacceptable periods.

4.5. Generic Data Structures

As described earlier the LSE also has a small set of generic data structure handling routines:

- **Pools** A pool of a certain type of data structures can be created, data units can be requested and returned to the pool, (re-) initialization routines can be specified to be called each time a data unit is returned to the pool.
- **Queues** A collection of generic single and double linked lists routines.
- **Plists** A list of data structures that reside in a pool linked by a list.

5. Abstract Network

At the basic to the design of the Group Communication Service is the strategy to take advantage of Local Area Network (LAN) technology, using different types of LANs like 8804-4 token-bus [ISO85a], 8802-5 token ring [ISO85b], FDDI [X3T86a] and Ethernet [ISO85c]. Although these LANs are quite different in their use of technology, one can determine a general set of properties that are to be offered by every LAN [Ver91a]. The *Abstract Network* is used to hide the LAN specific details from the protocol environment [Ruf91a], exporting a number of helpful (Table 2) properties that are used to implement the properties of the group communication protocols. These abstract network properties are partially provided by the LAN technology and is complemented by additional software.

The properties Pn1 and Pn2 guarantee detection of erroneous delivery by the LAN in case of the broadcast/multicast case. Properties Pn4 and

Pn1	*Broadcast:* Destinations receiving an uncorrupted frame transmission, receive the same frame.
Pn2	*Error detection:* Destinations detect any corruption by the network in a locally received frame.
Pn3	*Bounded omission degree:* In a network with *N* nodes, in a known interval, corresponding to *(k+1)* series of unordered transmissions, such that each of the *N* access points transmits one frame per series, all transmissions are indicated in all destination access points, in at least one series.
Pn4	*Full duplex:* Indication, at a destination access point, of frame reception, during transmission by the local source access point, may be provided, on request.
Pn5	*Network order:* Any two frames indicated in two different destination access points, are indicated in the same order.
Pn6	*Bounded transmission delay:* Every frame queued at a source access point, is transmitted by the network within a bounded delay.

Table 2: *Network Properties*

Pn5 are the foundation for the ordering properties of the group communication protocols. Pn3 and Pn6 define the behavior in the time domain, Pn3 denotes a *bounded omission degree*, based on failure detection and fault treatment, Pn6 depends on the particular network, its sizing, parameterizing and loading conditions. The *Abstract Network*, in a sense, extends the concept of LLC,† the LAN independent sublayer of the IEEE, and later ISO 802 standard [ISO85d].

5.1. Abstract Network Primitives

The user of the abstract network service has a number of primitives available for interaction with the network [Ruf91b].

- *Data Request primitives.* These primitives request the transmission of a frame.
  - ♦ **Group request** – multicast this frame to all members of a given group.
  - ♦ **Selective request** – multicast this frame to a given subset of the members of a group.
  - ♦ **Individual request** – send the frame to a specified station.
- *Data Receive primitives* – Indications of data from the network, or confirmations from the network interface if a given data request has been served or not.
- *Network Management primitives* – These primitives provide interaction with manageable objects in the abstract network like addresses, network sizing, load control, traffic monitoring, protocol characterization. All objects can be read, some can be set to new or predefined values.
- *Station Management primitives* – a number of primitives manipulate the stations presence on the network and the management of the multicast address space it will receive frames on.
  - ♦ **Stations** can be inserted or removed from the network. The routines initialize or shutdown the internal abstract network protocols and control the presence of the station on the network.
  - ♦ **Groups** can be opened and closed. This is the management of the multicast address space using either hardware or software selection.
  - ♦ **Selective** addresses can be set or removed. These are the identifiers used as the station selective address, used in subset addressing.
  - ♦ **Fault injection** mechanisms like making the station *deaf* or *dumb* to introduce faults for protocol testing.
- *Notification primitives* for flow control, network failure detection and station management.

5.2. Abstract Network Implementation

The Abstract Network presents the user with an interface to the real network network, through use of the primitives from the previous section. But not all network controller give the designer the same set of mechanisms to implement the Abstract Network, often additional software is needed to implement all properties of the Network correctly.

† Logical Link Control sublayer.

Implementations have been made for token-bus, token-ring and Ethernet, an experimental implementation for FDDI is in progress.

The abstract network is implemented independent from the *x*AMp protocol suite. Within SunOS the only abstract network implemented are of the Ethernet type (and LAN class, see [Ver92a]). Within each station a number of abstract networks instances is available to which a higher level protocol can connect, either directly from within the kernel or from user space through a device driver interface. This way the abstract network is not only available for the group communication service for can be used for other types of protocol development as well.

Binding of a protocol to an abstract network is done dynamically, and after this binding the abstract network instance is initialized to use a specified network interface (corresponding to its type, only Ethernet in the SunOS case) and to use a specified network type identifier (the protocol field in the Ethernet frame header).

An important aspect of the abstract network is the management of the multicast address space [Vog91a, Vog92a]. To all extend one should avoid using *broadcast* or *all-multicast* modes of the network controller, as one looses the advantage of hardware multicast address filtering. If this can not be avoided there are two possible schemes;

- All stations receive all messages from all other stations participating in the conversation, and address filtering is done by software.
- Messages are send using multiple point-to-point messages.

In both cases the real advantages of hardware multicast are nullified.

Per network interface a module handles the multicast address management for all abstract networks connected to that interface. For the Ethernet case there is a mapping between group identifiers and the multicast address, in contrast with the token-bus implementation made for the SPART/UE real-time environment where the *selective* address is part of the hardware address filtering scheme.

In the Ethernet version the selective address filtering is done by software. It are simple, inexpensive bit masking manipulations. Although the selective address for a particular abstract network can be altered, it is implicitly connected to the selective address assigned by the MGS to this station.

For each interface a number of statistics are kept to be able to identify load, sizing, error rate etc. This information is used to compute round trip estimates, omission timeouts, transmission delays, etc.

6. The Group Communication Protocol

The core of the Group Communication Service is the *eXtended Atomic Multicast Protocol* (*x*AMp) [Rod92a], which offers a number of qualities of service as described in Section 2 [Rod91a]. The selection of these QOS's was driven by user requirements put by diverse classes of distributed applications. These requirements arisen from the literature and largely from the needs of the group replication and membership protocols of Delta-4 architecture.

In the following Sections we describe the basic transmission procedure and its use by a number of the qualities of service.

```
0   // **tr−w−resp** ($m, ord, send, P_r, n_r, M_r$)
1   // "m" is a message to be sent. ($D_{(m)}$ is the set of recipients).
2   // "ord" is a boolean specifying if network order is relevant.
3   // "send" allows the first transmission to be skipped.
4   // $P_r$ is a set of processors from which a response is expected.
5   // $n_r$ is the number of responses expected.
6   // (usually $n_r = \#P_r$ ; $P_r = D_{(m)}$)
7   // $M_r$ is a bag of responses
8
9   $retries := 0;$
10  **do** // while
11      **if**($retries = 0 \mid ord$) **then**
12          $P_w := P_r$ ; $n_w := n_r$ ; $M_r := 0$ ;
13      **fi**
14      **if**($retries > 0 \mid send$) **then** $send(m)$; **fi**
15      $timeout := 0$; start a timer; // wait responses
16      **while** ($n_w > 0$ & $\neg$ timeout) **do**
17          **when** response ($r_m$) received from $p$ & $p \in P_r$ **do**
18              $add(r_m)$ to $M_r$;$n_w := n_w - 1$;
19              remove $p$ from $P_w$; **od**
20          **when** timer expires **do**
21              $timeout := 1$; **od**
22      **od**
23      $retries := retries + 1$;
24  **while**($retries < MAX$ & $n_w > 0$)
25
26  **if**($n_w > 0$) **then** check membership **fi**
```

Figure 2: *Transmission with response (tr-w-resp) procedure*

6.1. The Transmission with Response Procedure

Basic to the *x*AMp is the use of the abstract network service which offers an *unreliable multicast* service. In absence of faults the *broadcast* (Pn1), *full duplex* (Pn4) and *network order* (Pn5) properties of the abstract network provide message delivery at all connected stations in the same order. However, although errors can be considered rare in LANs, the occasional loss of messages – or omissions – cannot be prevented. Thus, the communication service must be able to recover from such errors. In the *x*AMp, omission errors are detected and recovered using a transmission with response procedure: it uses acknowledgments to confirm the reception of the message and detects omission errors based on the *bounded omission degree* property of the abstract network.†

The *tr-w-resp* procedure‡ is depicted in Figure 2. It consists of a loop, where the data message is sent over the network and responses are awaited for. The procedure waits during a pre-defined time interval for the responses (l.15), which are then inserted in a response bag (l.17) and exits when the desired number of responses is collected. If some responses are missing, the response bag is re-initialized (l.12) and the message re-transmitted. The main loop finishes when all the intended

† The detailed technique, as well as its advantages over other approaches such as diffusion based masking is discussed in detail in [Ver91a].

‡ It is a modified version of the procedure given in [Ver90a].

responses are received or when a pre-defined retry value is reached (l.23).

To preserve *network order*, the procedure re-transmits the message until it is acknowledged by all recipients in a same transmission. When order is not required, the procedure can be optimized by keeping responses in the bag from one re-transmission to the other (response messages are inserted only once in the response bag). For some omission patterns, this would allow the bag to be filled faster. To activate this mode, the flag **ord** must be set to false. Finally, the boolean variable **send** allows the user to specify that the message should be sent over the network on the first cycle of the procedure. This parameter is useful to allow another processors to collect responses – and execute the procedure – on behalf of the sender without immediately re-transmitting the message. In later sections we explain how this feature is used to provide some of *x*AMp qualities of service.

Several transmissions with response can be executing simultaneously, on the same or on different machines. We assume that messages can be uniquely identified. Different re-transmissions of the same message can also be distinguished. It is thus possible to route any response to the appropriate *tr-w-resp* instantiation (also called an *emitter-machine*).† To make a protocol tolerant to sender crashes, several emitter-machines may be activated concurrently, at recipients sites, for a same message transmission (in this case, responses must be also broadcasted). See *atLeast* agreement for an example.

6.2. *BestEffort* and *atLeast* Qualities of Service

A number of distributed applications do not need communication primitives that provide very strong order and agreement primitives, but do want to use the efficient dissemination of messages to a group of stations. To give support for this type of application demands the *x*AMp offers the *bestEffort* and *atLeast* primitives.

BestEffort is used to simply send a message to a group of stations. The user can specify the number of responses needed (n_r), or which named subset of addressees (P_r) needs to acknowledge the message. If the number of requested responses is zero the service is equivalent to an *unreliable multicast* service.

The *bestEffort* quality of service is not able to assure delivery in case of sender failure. In order to provide assured delivery, in the presence of sender failures, we make every recipient responsible for the termination of the protocol. In consequence, *tr-w-resp* is invoked both at the sender and at the recipients, as depicted in the figure (l.7). However, to avoid superfluous re-transmissions of the data message, recipients skip the first step of the *tr-w-resp* procedure, using the **send** boolean parameter). In the no fault case, the data message will be acknowledged by all intended recipients, these acknowledgments will be seen by the all the participants and no retransmission takes place.‡ As with *bestEffort* several variants on agreement are possible by choosing the set of stations that need to respond (P_r) or the number of responses (n_r) needed.

† Since several emitter-machines can run in parallel, the protocol implementation is able to execute several user requests at the same time. However, since a node usually has limited resources (memory and cpu), the implementation may restrict the number of simultaneous transmissions, for instance keeping a fixed size pool of emitter machines. Some qualities of service may impose additional restrictions on parallelism.

‡ This algorithm can be improved to avoid multiple retransmissions when a single omission occurs, by making the recipients use slightly different timeout values, and making the protocol refraining from re-sending when a retransmission from other participant is detected before the timeout expires.

If the number of responses needed is smaller then the set of addressed stations ($n_r \leq \#D_{(m)}$), the primitive will assure that *at least* that number of the addressees receive the frame even if the sender fails. This is satisfactory to implement quorum based protocols.

In the case where the number of responses required is equal to the number of members of the group, the primitive is also called *reliable* multicast. Reliable multicast is used as the base of two other qualities of service: *causal* and *delta*.

6.3. The Atomic Quality of Service

The *atomic* quality of service, in relation to the other qualities of service previously described, introduces the assurance of total order. This can be achieved exploiting the properties of the abstract network: in fact messages are naturally ordered as they cross the LAN medium (abstract network property Pn5). To preserve network order, a mechanism must be implemented to ensure that the messages are delivered to the user respecting the order they have crossed the network and, when a message crosses the network several times, that a unique re-transmission is used to establish this order. This requires extra work both at the sender and at the recipient sides, as described below.

In each recipient, is maintained a *reception queue*, where messages are inserted by the order they cross the network. Since at the moment of reception, a recipient as no way to know if the message was also received by the other recipients, the message cannot be delivered immediately to the user. Instead, it is stamped as *unaccepted* and kept in the queue until there is an assurance that it was inserted in the same relative position in *all* recipient's queues. If meanwhile, a re-transmission is received, the message is moved to the end of the queue. On its side, the sender invokes *tr-w-resp* activating the "ord" flag, thus requiring the re-transmission of the message until all recipients acknowledge the same retry. When a successful re-transmission is detected, the sender issues an *accept* frame, committing the message. When the accept frame is received, the recipients mark the associated message as *accepted* and deliver it as soon as it reaches the top of the queue.

If a receiver is not able to process the message[†] due to lack of resources like buffer space or scheduling guarantees it notifies the sender by returning a *not-ok acknowledgement* to the sender. The sender reacts on the receipt of such a negative acknowledgement by issuing a *reject* instead of an *accept* message. Upon receipt of the reject all recipients discard the corresponding data message.

The atomic service consists of a **two-phase accept** protocol (see Figure 3) that resembles a commit protocol where the *sender* coordinates the protocol: In the dissemination phase the data message is sent to all recipients, who have to respond if they will be able to process the message. In the second phase (decision phase) the sender decides to send either an *accept* or a *reject* message. To increase performance the accept message is sent using a *negative* acknowledgement scheme: If a recipient has not received a decision message due to an omission, it will detect this through a timeout mechanism and send a *Request-Decision* frame. Using this scheme a second round of acknowledgements is avoided increasing the performance. In the scenario where there is an omission of an *accept* message, termination of the protocol

[†] More related work on *inaccessibility* can be found in [Ruf92a, Ruf92b].

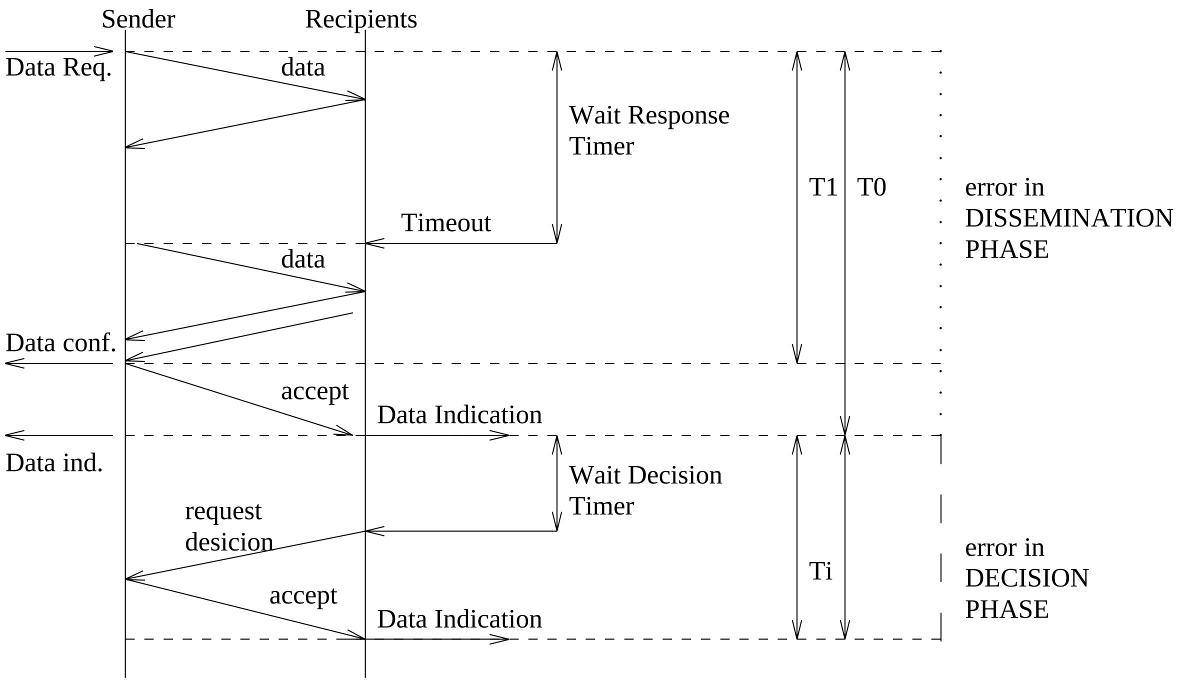

Figure 3: *Structure of the multi-phase Atomic Multicast protocol*

is delayed but due to the low error rate expected in local area networks, throughput is significantly improved.

Since in the two-phase accept the sender coordinates the protocol, some exception mechanism must be implemented to overcome its failure. In the *atomic* quality of service, protocol execution is carried on, in the event of sender failure, by a termination protocol. This termination protocol is executed by an *atomic monitor* function. There is no permanent monitor activity however – so to speak, a monitor only exists when needed. The monitor impersonates the failed sender but never re-transmits a data message on its behalf. It just collects information about the state of the transmission and disseminates an decision (reject or accept) accordingly [Ver90b].

7. Clock Synchronization

A number of classes of distributed applications require access to a global time base, for implementation of coordinated decentralized actions in the time domain, sensoring, performance measurement or timestamping of events.

It is possible to provide such a timebase by using a centralized time service, resident in a single node of the system. This solution is not fault-tolerant, exhibits poor performance if clocks need to be frequently read, and errors are introduced due to variation of transmission delays. The common solution for the clock synchronization problem lies on using the processor hardware clock to create a virtual clock at each node, which is locally read. All virtual clocks are synchronized by a *clock synchronization algorithm.* Surveys of existing clock synchronization algorithms can be found in [Sch87a, Ram90a, Kop89a]. Of the available software algorithms the convergence-non-averaging algorithms are attractive because they use the convergence function both to generate the re-synchronization event and to adjust virtual clocks.

However, the existing algorithms of this class have a major disadvantage: the precision of their convergence function is limited by the maximum message transmission delay in the system.

Verissimo and Rodrigues [Ver92b] have developed an algorithm which overcomes the limitation caused by the uncertain message delays, by using the properties of broadcast networks. The algorithm is implemented within the Group Communication Service using dedicated services available to the *x*AMp protocol suite [Rod91b].

The global time is available to the user through library functions that read the virtual clock value.

8. Processor Management

To provide efficient management of stations, a low-level processor membership protocol [Rod92b] is developed that deals with availability information on the nodes in the network. This information is not static: during the lifetime of the system, stations will join, leave and, possibly, fail. The protocol runs directly on top of the LAN (Abstract Network) to achieve improved performance and to offer a service that can be used by other protocol layers.

The processor membership has two major goals:

- It keeps a complete, and updated, list of a selected group of stations, participating in the multicast traffic (target systems typically include up to 32 nodes). This group is called the *Multicast Group of Stations* or simply MGS. The MGS protocol assures that the membership view is updated consistently in the presence of joins, leaves and failures. Changes in the MGS membership are indicated to the protocol users.
- It implements a mapping function that translates unique node identifiers into short-addresses. To enable run-time reconfiguration, the mapping is not statically pre-defined and new stations are able to, at any time, obtain a short-address. This mapping is *universal* and *stable*, meaning that, in all stations, the same short-address corresponds to the same station and that correspondence remains unchanged during the lifetime of the system.

The use of short addresses, as also exploited in Autonet [Sch90a], is to provide fast address manipulation based on bitmasking. These operations provide a significant performance improvement and, when the maximum number of multicast stations is small, allows the recognition of selective addresses to be implemented in hardware, by the chipset of the underlying network.†

8.1. Protocol Service

Our group membership protocol provides the mapping function referred above by maintaining a table with information about all stations participating in the multicast traffic. For efficiency and fault-tolerance, the table is replicated at every group member. The table includes an array of state entries, each entry storing information about a given member of the group: an entry contains, at least, the node unique identifier and a boolean stating if the node is alive. Additionally, the

† For instance, the MC68824 token-bus controller has a *group address mask* which can be set to filter messages in function of a bit value.

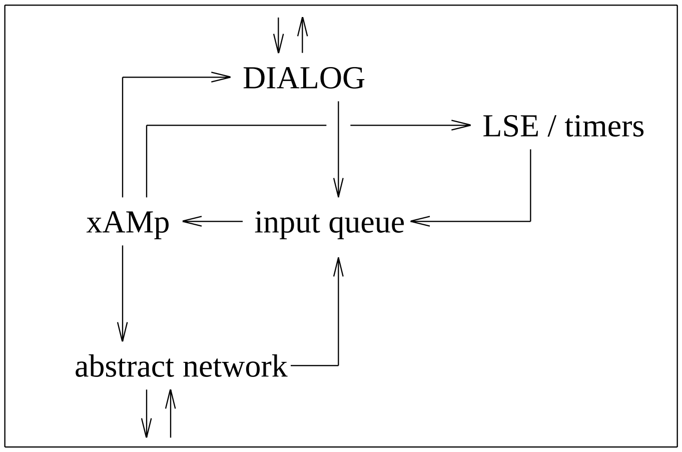

Figure 4: *Interaction between several modules*

entry may store user related data. The short-address associated with each MGS member is stored implicitly: it corresponds to the index of the associated entry in the table.

A station may be connected to the network without participating in the group membership protocol. In order to join the MGS group it must execute a *MgsJoin* operation. The join operation requires exchange of messages with the other MGS members to acquire the state table, insert itself and obtain a short-address. Upon an insertion in the MGS, a station is informed of any change in the MGS membership by an *MgsChange* indication. A station may leave the MGS by executing an *MgsLeave* operation. The MGS membership is checked at every execution of a Join or Leave or when a specific *MgsCheck* operation is explicitly invoked. The *MgsCheck* can be called periodically or upon the detection of an event that raises suspicion about the failure of a MGS member.

When a station joins the MGS, it acquires a short-address which will remain associated with that station. Even if the station fails or leaves the MGS group, the short-address remains assigned to the station, such that the remaining stations can refer to it by the associated short-address. If the station recovers and executes a new join, it obtains its old short address. A dedicated operation, *MgsDelete* is used to remove a station from the MGS table and to release the associated short-address. Since there is a local copy of the MGS table available at every station, translation between unique identifiers and short-address is a purely local operation.

Once the MGS protocol has inserted the station into the group it makes use of the *x*AMp primitives to assure the detection of failed stations. The MGS protocol joins an *x*AMp group that includes all available stations, within this group *keep-alive* message are sent to trigger the *Group Monitor* in case of failure of a station. The Group Monitor will automatically call the MGS protocol primitives to assure a consistent view of the available stations.

9. **Dialog**

One of the goals in the design of the UNIX kernel version of the Group Support Service was that the service should be available through the standard UNIX network interfaces like *streams* and *sockets*. Although the structure of both BSD and System V style network protocols didn't match the structure of the *x*AMp protocol core we wanted to make an effort of offering the service through these interfaces. The interface between the *x*AMp and the socket and streams endpoint environment is named **Dialog**.

The *x*AMp is structured as a state machine with an input channel on which messages from user, network and synchronous timers arrive and a collection of routines that are called as result of changes in the state machine, resulting in confirmation and indication messages to the user, interaction with the abstract network, including sending of messages, and manipulation of timers (see Figure 4).

The *x*AMp runs as *light weight process* (thread) in the kernel, sleeping on the input queue channel. If a message is placed in the input queue a wakeup of the *x*AMp thread is generated. If the *x*AMp outputs messages to the user it calls a routine from a predefined collection of *dialog* routines. These routines handle the two types of user messages generated by the *x*AMp:

- **Confirmation**: The requested operation has completed. A confirmation can be positive or negative regarding the result of the operation.
- **Indication**: data output is produced for the user, this could be a new group or processor view, a time synchronization message or data received for a group member.

The user can specify through control operations which types of confirmations and indications he does want to receive.

Confirmations are necessary as the caller is not blocked in the submitting routine until the operation is successful. For the socket version this requires some additional mechanisms in the dialog module to maintain the blocking semantics of the *send* and *write* system calls.

The Dialog module for the *streams* version was expected to link up better with the *x*AMp, as *streams* are also model after a submit/confirmation/indication model. Already in the prototype phase it became noticeable that using streams in SunOS is a very expensive method of designing network software, it added almost a 80 msec overhead for interaction with the abstract network driver. This scale of delay was unacceptable for our goals, and we stopped with the development of the streams driver.

As an alternative to the streams environment we built a *Dialog* interface to a regular UNIX device driver that has the same user semantics as a streams driver. This implementation turned out to have good performance with low overhead, as all message handling is tuned for this specific environment.

The driver, as well as the socket code, supports all UNIX type operations like select, signals, non-blocking read, etc.

10. Formal Specification and Verification

There is now a general agreement that protocols must be validated. We have chosen to do a formal design specification (as opposed to simulation) because this will give you insight in possible errors in your protocol design. As an approach to formal verification we decided to use model checking instead of deductive proof methods for the same reason: it is of great help for the detection of errors. In order to apply these techniques, one needs the description of a complete system consisting of a fixed number of communicating entities and their interaction environment. Such a complete system is called a *scenario* Practically, validation comes down to the construction of a certain number of critical scenarios and their formal verification by using a tool.

For the verification of AMp we have used the verification tool Xesar [Ric87a, Gra89a]. This tool evaluates properties given by formulas of temporal logic on a *model* generated from a scenario to be verified. The *model* represents the complete state graph obtained automatically from a scenario written in Estelle/R, a variant of Estelle (communication is modeled by *rendez-vous*). The basis for the verification is the complete Estelle/R design specification of the AMp. Since a *closed system* (for each message, the sender and the receiver must be described) is needed for the verification, a description of the environment is also needed, i.e. the modeling of the adjacent protocol layers: the network layer and the user layer.

The work on the verification of the AMp has been very successful, a number of possible errors has been found, and the results of the verification have given us great confidence in the correctness of the protocol [Bap90a].

The implementation was also subject to a validation effort: a fault injection campaign is in course, with the aim of forecasting faults and assisting in its removal, with the help of a specialized tool [Arl90a].

11. Performance Measurement

Throughout this paper we have stated that achieving a responsive service was one of the main goals to achieve. In this section we will describe some of the performance measurements we have executed. The next section will focus on comparing these results with those of comparable systems.

The measurements have been performed in two different operating system environments:

- **SunOS 4.1.1** – running on SPARCstations I, IPC's and SLC's
- **Mach2.5** – running on 33Mhz i486 machines of Taiwanese origin.

The environments differ most in the implementation of the *Abstract Network*. We did not have the source code for the SunOS operating system available and used the *ether_family* mechanism to insert are protocols in the de-multiplexing process. For the Mach2.5 port we were able to implement the Abstract Network exactly as we designed, having access to all functionality of the lowest layers.

Measurements were done by using the special performance device driver, which allows us to make timestamps at different stages of the frame manipulation process, collecting these timestamps afterwards.

The main application of the Group Communication Service is in the area of responsive and real-time systems. In this context we are more interested in the timely execution of the primitives and in the exceptions in the execution times. We recognize the importance of throughput of large batches of messages, but the protocols are tuned towards single message handling and guaranteed timely termination of the protocols.

The first set of measurements are to determine the latency caused by the Abstract Network and the physical transport over the network. Relevant is the size of the buffer used. The number of stations used is not

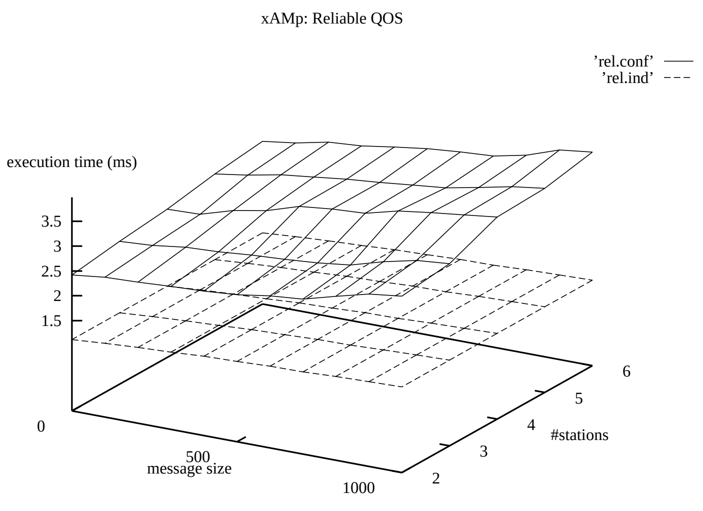

Figure 5: *Performance of the reliable quality of service*

of any influence as all message are transmitted using hardware multicast.

When selecting only the i486 machines the latency dropped significantly with 30 to 35%, this is caused by the more optimal abstract network implementation.

The variance of the large buffer transfers is larger because of the collisions on the network. When we repeated the tests on an isolated Ethernet and the variance approaches that of the smaller buffers.

The second series of tests involved measurements of the *bestEffort*, *atLeast* and *reliable* primitives. As these primitives involve exchanges of acknowledgements the number of stations plays a role in the performance of the protocols. Two points of measurement are taken:

1. The moment the data is indicated to the user.
2. The moment the sender is confirmed of the termination of the protocol.

All three primitives are not concerned with ordering properties and indicate the data as soon at it arrives at the Group Communication Service. The protocols terminate after the requested number of the specified subset of group members have acknowledged the message (see Figure 5).

Abstract network roundtrip time (msec)		
frame size (bytes)	i486 & SPARC stations	i486 stations
1	1.04	0.78
100	1.13	0.82
200	1.27	0.87
500	1.32	0.93
1000	1.39	1.01
1450	1.48	1.12

Table 3: *Abstract network latency*

Group membership primitives						
# stations	0	1	2	3	4	5
join	2.5	3.2	4.1	4.9	5.6	6.8
leave	1.2	2.0	2.4	2.6	2.9	3.2

Table 4: *group join performance*

In case of the *causal* ordered primitive, there is a small overhead for handling of the ordering protocol which resides on top of the reliable primitive. But we have noticed that in the case when all related messages already have been received the overhead is in the order of 170 microseconds.

The *atomic* primitive is a **two-phase accept** protocol (see Figure 3) that confirms the user about the result of the operating after the *dissemination* phase and indicates the data after the *decision* phase (see Figure 6).

As the last results we want to report on the performance of the group membership primitives. In Table 4 the costs of joining and leaving a group is presented.

12. **Performance Comparison**

From all published research in the area of group communication we will discuss our results in comparison with the results of ISIS, Amoeba, and Consul/Psync. We have chosen these three systems because they all represent a different main stream in group communication.

The *ISIS* [Bir91b] toolkit offers a versatile set of communication primitives combined with higher level implementations of distributed algorithms. The ISIS protocols run in user space using the standard communication channels. The authors have put the emphasis on throughput sacrificing some of the responsive behavior. As the toolkit relies on the

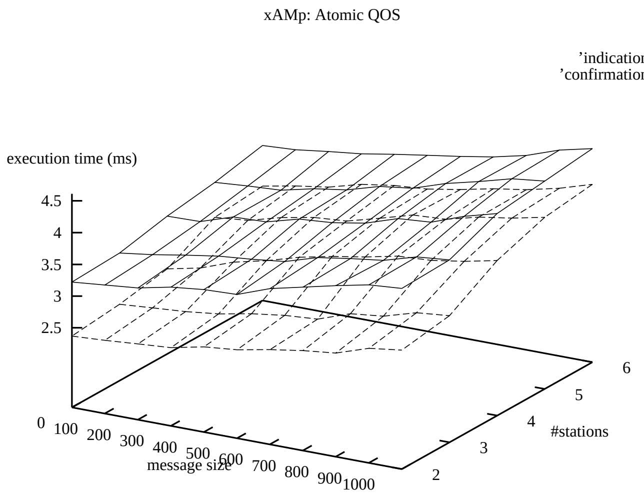

Figure 6: *Performance of the atomic quality of service*

standard Internet protocols to transport their messages they are not bound by the scope of a Local Area Network, but do have to deal with the sometimes unpredictable behavior of the UDP/TCP/IP layers. Obviously these transport methods make the toolkit very portable but a significant performance penalty is paid to achieve this.

The ISIS CBCAST primitive is comparable with the reliable/causal primitive offered by the *x*AMp protocol suite. The ABCAST can be compared with the atomic two phase accept protocol described earlier, although ABCAST is not able to distinguish *inaccessibility* from communication or processor failure. Although the complexity of the protocols is comparable the performance of the *x*AMp primitives is much better, a 0 bytes CBCAST (6 stations) takes about 17.8 msec, while the *x*AMp causal primitive takes 2.96 msec (confirmation). For 1K packet the costs are 21.1 and 3.9 msec respectively.

The performance analyses given in [Bir91c] show that more than 75% of the measured latency in the ISIS system is caused by the operating system layers. Our Group Communication Service gains in performance by locating its service as close to the network as possible, bypassing as much system layers as possible. Another reason for the improved performance is the use of hardware multicast by the abstract network, minimizing the message traffic. At Cornell a total redesign of ISIS is in progress which will result in a system that will approach the perform of the *x*AMp primitives.

In *Amoeba* [Tan90a, Ren88a] the group communication service offers only one primitive: total order within a group [Kaa89a]. The protocol uses sequencer sites to regulate the order of the messages. The protocols simpleness results in very high performance but lacks the ability to be used in more complex environments with different application requirements. Some of the major criticisms are the inability of the protocol to support overlapping groups and the lack of timely omission detection. Recently the protocol has been adjusted to make use of the Fast Local Internet Protocol (FLIP) [Kaa92a] which provides reliable multicasting. Ongoing research at the Vrije Universiteit is focused on extending the current protocols.

When comparing performance figures it is clear that the complexness of our service makes it not competitive with the less versatile service of Amoeba. On a lightly loaded Ethernet the Amoeba protocol makes an atomic broadcast to 10 station within 1.5 msec. There are no acknowledgements involved, and there is little overhead from the network interface modules.

Consul [Mis92a] is a communication substrate for building fault-tolerant systems, the system relies heavily on the services offered by Psync, a group communication protocol designed to preserve causality based on the use of a context graph. The systems are build within the *x*-kernel [Hut90a], a protocol development environment from the University of Arizona. The performance of this dedicated environment is almost comparable to our implementation, both causal and total order service have a latency that is 0.5 to 1.5 msec higher than the *x*AMp protocols. We believe the slightly worse performance of the total order service can be related to the fact that this service is build on top of the causal order primitive.

13. **Lessons Learned**

Some conclusion from the practical side are:

- Having multiple qualities of service helps the builder of distributed applications to minimize communication cost.
- Exploiting network properties makes building of group communication easier and more responsive.
- Integration inside the operating system has yielded good performing service.
- Integration into a operating system without having access to the source code should be avoided.
- Implementing responsive protocols using SunOS streams is not possible.
- Using formal verification techniques has improved the confidence in the correctness of the protocols.
- The different addressing modes have improved the usefulness of the service.
- The short addresses have improved the address manipulation enormously but do not scale well.
- Portability is possible even within between operating system code if at design time an effort has been made to locate system dependencies.

The experiences with the design of the group support service form the basis of a report on requirements for building group support systems, see [Vog92a].

14. **Future Directions**

Our current research continues to focus on integration of the group concept in different areas of distributed computing. Our main goal is to achieve a high performance group service that can be used in real-time and responsive systems. We will also focus on how to build responsive group support for large scale distributed systems, especially in the area of CSCW. Another main line is the development of group management protocols [Ver92a].

In the *Navigators* project we are focusing on a total redesign of the *x*AMp protocol suite to to incorporate new ideas on responsive systems, dedicated support by micro-kernels, low-level high-performance transport mechanisms, multi-level failure detectors, etc. In the same environment we try to incorporate internetworking support for group communication at MAN and WAN scale [Vog91b].

Our new environment is being developed for the Mach 3.0 microkernel and a prototype is planned for the end of 1992. Also cooperation between newly designed ISIS modules and Navigators protocols are foreseen.

Formal verification and specification techniques will be more integrated into the design process as they have shown in our case to improve the quality of the protocols built.

15. Summary

In this paper we have presented the implementation of a Group Communication Service aimed at achieving high-performance to support distributed applications that have responsive requirements. Scalability has been traded for timely protocol execution of the protocols.

The main concepts are described as well as the actual implementation, and design decisions have been motivated. For more details on specific parts of the service the reader is referred to [Ver90b, Rod92a].

We have described the performance of our protocols and compared these to three other popular group communication services. When looking at the performance figures is becomes clear that the different protocols that form the core of our service can compete with any other know group communication system in both performance and quality of the offered services.

Acknowledgements

Most of this work has been developed in the scope of the Delta-4 architecture. We wish to thank M. Baptista, A. Casimiro, M. Chereque, J. Etheringhton, H. Fonseca, B. Guilmore, R. Ribot, C. Rodriguez, and J. Rufino for their valuable contribution to the evolution and engineering of *x*AMp. The authors are indebted to several other people in the Delta-4 project for the many criticisms and suggestions made during the design and implementation of *x*AMp, particularly to P. Bond, D. Powell, JL. Richier, D. Seaton and J. Voiron. We also thank the fruitful exchanges of ideas with Ken Birman, Larry Peterson and Rick Schlichting.

References

[Arl90a] J. Arlat, M. Aguera, Y. Crouzet, J. Fabre, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: a Methodology and some Applications," *IEEE Transactions on Software Engineering*, IEEE (February 1990). Special Issue of Experimental C.Sc.

[Bap90a] M. Baptista, L. Rodrigues, P. Veríssimo, S. Graf, J. L. Richier, C. Rodriguez, and J. Voiron, "Formal Specification and Verification of a Network Independent Atomic Multicast Protocol," *Third International Conference on Formal Description Techniques (FORTE 90)*, Madrid, Spain, IFIP (November 1990).

[Ber89a] Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska, and Henry M. Levy, "Lightweight Remote Procedure Call," *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pp. 102-113, ACM (1989).

[Bir91c] Kenneth Birman, Andre Schiper, and Pat Stephenson, "Lightweight causal and Atomic Group Multicast," *ACM Transactions on Computer Systems* **9**(3) (August 1991).

[Bir91a] Kenneth P. Birman, "The Process Group Approach to Reliable Distributed Computing.," TR 91-1216, Cornell University, Ithaca, USA (July 1991).

[Bir91b] Kenneth P. Birman, R. Cooper, and B. Gleeson, “Design Alternatives for Process Group Membership and Multicast,” TR91-1185, Cornell University, Ithaca, USA (december 1991).

[Bir84a] Andrew D. Birrell and Bruce Jay Nelson, “Implementing Remote Procedure Calls,” *ACM Transactions on Computer Systems* **2**(1) (February 1984).

[Che85a] D. Cheriton and W. Zwaenepoel, “Distributed Process Groups in the V-Kernel,” *ACM Transactions on Computer Systems* **3**(2) (May 1985).

[Che88a] Greg Chesson, “XTP/PE Overview,” *13th Local Computer Network Conference*, Minneapolis-USA (October 1988).

[Coo85a] Eric C. Cooper, “Replicated Distributed Programs,” *10th ACM Symposium on Operating Systems Principles*, Berkeley, California 94720, USA, ACM (November 1985).

[Cri90a] Flaviu Cristian, Robert D. Dancey, and Jon Dehn, “Fault-Tolerance in the Advanced Automation System,” *Digest of Papers, The 20th International Symposium on Fault-Tolerant Computing*, Newcastle-UK, IEEE (June 1990).

[Dru92a] Peter Druschel and Larry L. Peterson, “High-Performance Cross-Domain Data Transfer,” TR 92-11, University of Arizona, Tucson, USA (March 1992).

[Fon90a] H. Fonseca, L. Rodrigues, J. Rufino, and P. Verı́ssimo, “Local Support Environment: User Specification,” RT/50-90, INESC, Lisboa, Portugal (August 1990).

[Gar89a] H. Garcia-Molina and Annemarie Spauste, “Message Ordering in a Multicast Environment,” *9th Internacional Conference on Distributed Computing Systems*, pp. 354-361, IEEE (June 1989).

[Gra89a] S. Graf, J. L. Richier, C. Rodriguez, and J. Voiron, “What are the Limits of Model Checking Methods for the Verification of Real Life Protocols?,” pp. 275-285 in *Automatic Verification Methods for Finite State Systems*, ed. J. Sifakis, Springer-Verlag (June 1989).

[Her89a] A. J. Herbert, J. Monk, and R. van der Linden, *The ANSA Reference Manual*, Architecture Projects Management, Ltd, Cambridge, UK (July 1989).

[Hut89a] Norman C. Hutchinson, Shivakant Mishra, LArry L. Peterson, and Vicraj T. Thomas, “Tools for Implementing Network Protocols,” *Software – Practice and Experience* (September 1989).

[Hut90a] Norman C. Hutchinson and Larry L. Peterson, *The x-Kernel: An Architecture for Implementing Network Protocols*, University of Arizona, Tucson, USA (1990).

[ISO85a] ISO, “Token Passing Bus Access Method,” DIS 8802/4-85 (1985).

[ISO85b] ISO, “Token Ring Access Method,” DP 8802/5-85 (1985).

[ISO85c] ISO, “Carrier Sense Multiple Access with Collision Detection,” DIS 8802/3-85, ISO (1985).

[ISO85d] ISO, “Logical Link Control,” DIS 8802/2-85, ISO (1985).

[Kaa89a] Frans M. Kaashoek, Andrew S. Tanenbaum, Susan Flynn Hummel, and Henri E. Bal, “An Efficient Reliable Broadcast Protocol," *ACM Operatings Systems Review*, pp. 5-19 (October 1989).

[Kaa92a] Frans M. Kaashoek, Robert van Renesse, Hans van Staveren, and Andrew S. Tanenbaum, "FLIP: an Internetwork Protocol for Supporting Distributed Systems," *ACM Transactions on Computer Systems* (1992).

[Kop89a] H. Kopetz, G. Grunsteidl, and J. Reisinger, "Fault-tolerant Membership Service in a Synchronous Distributed Real-time System," *Int. Working Conference on Dependable Computing for Critical Applications*, Sta Barbara – USA, IFIP WG10.4 (August 1989).

[Mis92a] Shivakant Mishra, Larry L. Peterson, and Richard Schlichting, *Consul: A Communication Substrate for Fault-Tolerant Distributed Programs,* University of Arizona, Tucson, USA (1992).

[Pet89a] Larry L. Peterson, Nick C. Buchholdz, and Richard D. Schlichting, "Preserving and Using Context Information in Interprocess Communication," *ACM Transactions on Computer Systems* **7**(3) (August 1989).

[Pow91a] D. Powell, *Delta-4 – A Generic Architecture for Dependable Distributed Computing,* Springer Verlag (November 1991).

[Ram90a] Parameswaran Ramanathan, Kang G. Shin, and Ricky W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *Computer*, pp. 33-42, IEEE (October 1990).

[Ren88a] Robert van Renesse, Hans van Staveren, and Andrew S. Tanenbaum, "The Performance of the Worlds's Fastest Distributed Operating System," *ACM Operatings Systems Review*, pp. 25-34 (October 1988).

[Ric87a] J. L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron, *XESAR: a Tool for Protocol Validation – User Manual,* Laboratoire de Genie Informatique, Grenoble, France (1987).

[Rod91b] L. Rodrigues, P. Veríssimo, and A. Casimiro, "xAMp Time Service Implementation Specification," RT/-91, Delta-4 Project, INESC, Lisboa, Portugal (October 1991).

[Rod92a] L. Rodrigues and P. Veríssimo, "*x*AMp: a Multi-primitive Group Communications Service," *11th Symposium on Reliable Distributed Systems*, Houston, Texas, IEEE (October 1992).

[Rod92b] L. Rodrigues, P. Veríssimo, and J. Rufino, "A low-level processor group membership protocol for LANS," RT/-92, INESC, Lisboa, Portugal (1992).

[Rod91a] Luís Rodrigues and Paulo Veríssimo, "*x*AMp: A Versatile Group Communications Service," *ERCIM Workshop on Distributed Systems*, Lisboa, Portugal (November 1991).

[Ruf92b] Jose Rufino and Paulo Veríssimo, "Minimizing token-bus inaccessibility through network planning and parameterizing," *EFOC/LAN92 Conference*, Paris, France, IGI (June 1992).

[Ruf91a] J. Rufino, P. Veríssimo, and L. Rodrigues, "Abstract Network Specification," RT/-91, INESC, Lisboa, Portugal (October 1991).

[Ruf91b] J. Rufino and P. Veríssimo., “Design Requirements of the Abstract Network User Interface,” RT/-91, INESC, Lisboa, Portugal (January 1991).

[Ruf92a] J. Rufino and P. Veríssimo, “A study on the inaccessibility characteristics of ISO 8802/4 Token-Bus LANs,” *IEEE INFOCOM’92 Conference on Computer Communications*, Florence, Italy, IEEE (May 1992).

[Sch87a] Fred B. Schneider, *Understanding Protocols for Byzantine Clock Synchronization*, Cornell University, Ithaca, New York (October 1987).

[Sch89a] Michael D. Schroeder and Micheal Burrows, “Performance of Firefly RPC,” *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pp. 83-90, ACM (1989).

[Sch90a] Michael D. Schroeder, Andrew D. Birrell, Micheal Burrows, Edwin H. Satterthwaite, and Charles P. Thacker, “Autonet: a High-Speed, Self-configuring, Local Area Network Using Point-to-point Links,” 59, Digital, Systems Research Center, Palo Alto, California (April 1990).

[Tan90a] Andrew S. Tanenbaum, Robert van Renesse, Hans van Staveren, G. J. Sharp, Sape J. Mullender, Jack Jansen, and Guido van Rossum, “Experiences with the Amoeba Distributed Operating System,” *Communications of the ACM* (December 1990).

[Ver90b] Paulo Veríssimo, “Group Communications Support,” in *Delta-4 A Generic Architecture for Dependable Distributed Computing*, ed. D. Powell, Springer Verlag (1990).

[Ver89a] P. Veríssimo and L. Rodrigues, “Order and Synchronism Properties of Reliable Broadcast Protocols,” RT/66-89, INESC, Lisboa, Portugal (December 1989).

[Ver90a] P. Veríssimo and J. A. Marques, “Reliable Broadcast for Fault-Tolerance on Local Computer Networks,” *Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama, USA, IEEE (Oct 1990).

[Ver91a] P. Veríssimo, J. Rufino, and L. Rodrigues, “Enforcing Real-Time behaviour of LAN-based protocols,” *10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, IFAC (September 1991).

[Ver92a] P. Veríssimo and L. Rodrigues, “Group Orientation: a Paradigm for Distributed Systems of the Nineties,” *3rd Workshop on Future Trends of Distributed Computing Systems*, Taipe, Taiwan (April 1992).

[Ver92b] P. Veríssimo and L. Rodrigues, “A posteriori Agreement for Fault-Tolerant Clock Synchronization on Broadcast Networks,” in *Digest of Papers, The 22th International Symposium on Fault-Tolerant Computing* (July 1992).

[Vog91a] Werner Vogels and Paulo Veríssimo, “Process Groups and reliable multicast communication in extended LANs, MANs and Internetworks,” RT/-91, INESC, Lisboa, Portugal (August 1991).

[Vog91b] Werner Vogels and Paulo Veríssimo, “Supporting Process Groups in Internetworks with Lightweight Reliable Multicast Protocols," *ERCIM Workshop on Distributed Systems*, Lisboa, Portugal (November 1991).

[Vog92a] Werner Vogels, Luís Rodrigues, and Paulo Veríssimo, "Requirments for High-Performance Group Support.," *5th ACM SIGOPS European workshop*, Mont Saint-Michel, ACM (September 1992).

[X3T86a] X3T9.5, "FDDI documents: Media Access Layer, Physical and Medium Dependent Layer, Station Mgt.," FDDI, X3T9.5 (1986).