xAMp: A Protocol Suite for Group Communication

INESC Technical Report RT/___-92

L. Rodrigues, P. Veríssimo

January 1992

xAMp: A Protocol Suite for Group Communivation[*]

Luís Rodrigues, Paulo Veríssimo
INESC[†]
Technical University of Lisboa
e-mail:... ler@inesc.pt, paulov@inesc.pt

Abstract

The xAMp is a highly versatile group communications service aimed at supporting the development of distributed applications, with different dependability, functionality, and performance requirements. This paper describes the services provided by xAMp and the protocols used to implement them. These range from unreliable and non-ordered to atomic multicast, and are enhanced by efficient group addressing and management support. The basic protocols are synchronous, clock-less and designed to be used over broadcast local-area networks, and portable to a number of them. The functionality provided yields a reasonably complete solution to the problem of reliable group communication.

Whilst other protocols exist that offer similar services, we follow a new engineering approach by deriving all qualities of service from a single basic procedure. Thus, their implementation shares data structures, procedures, failure-recovery algorithms and group monitor services, resulting in an highly integrated package.

1 Introduction

Distributed systems are widely used today, encouraging the development of applications that require progressively more of the distribution support. Well-known styles of distributed computing such as RPC and client-servers, are very useful and mature. Its widespread use was in fact necessary to show that its point-to-point and request-with-reply nature does not provide an universal solution to the increasing demands of developers of distributed applications, requiring complex and highly concurrent interactions between several participants, whose membership may be largely dynamic. Striking examples are found in the domains of computer supported cooperative work, and distributed computer control. These and other examples exemplified in the literature [14,18,5], require complementing paradigms.

One such paradigm gaining increasing acceptance is *reliable group communication (multicasting)*, concerning the dissemination of information to a *group* of participants in a system. The implementation of this paradigm meets a number of problems, due to natural impairments of the networking machinery: multicasted information can be lost or corrupted; may reach only a subset of the intended recipients; partitions may occur, leaving the recipients isolated, at least temporarily. Even when the information is not lost, it may be delivered in arbitrary order or at an arbitrary time, whereas the user might have expected a given ordering or timeliness.

Algorithms and protocols to solve these problems have been presented in the last years. They have been named after the several flavors provided: atomic, ordered, causal, reliable, etc.

[*]A shorter version of this report was published in the Proceedings of the 11th Symposium On Reliable Distributed Systems, Oct, 1992 Houston, Texas, © 1992 IEEE

[†]Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6º - 1000 Lisboa - Portugal, Tel.+351-1-3155150, Fax+351-1-525843. This work has been partially supported by the CEC, through Esprit Project 1226 - DELTA-4.

There are systems built according to these principles, using one or more of these protocols, such as the ISIS group toolbox [5], the group availability services of the IBM AAS [13], the conversational group support of PSYNC [24], the group membership and replication management services of the DELTA-4 distributed fault-tolerant architecture [25]. Under a system perspective, a systematics of *group-orientation* is developing. It should lead, in a top-down approach, to the definition of group communication and management services in response to pre-defined user requirements: support of application (or problem) classes and group types; required properties of agreement, order and synchronism; naming and addressing; time and value domain correctness (fault-tolerance, real-time).

In this paper we present an attempt in that direction, the *x*AMp, a *multi-primitive group communications service*. The *x*AMp is a complete redesign of its predecessor, the AMp [31], used as the communications support of the DELTA-4 system[1]. *x*AMp is aimed at supporting the development of distributed applications, with different dependability, functionality, and performance requirements. *x*AMp attributes emerged from the lessons learnt by experimenting with AMp in DELTA-4 in the past few years: *versatility*: a range of qualities of service; *homogeneity*: single core for a multi-protocol structure; *efficient name-to-address translation support*: logical group and sub-group addressing, selective physical addressing; *knowledge about group participants*: separation of membership management (user-oriented) from monitoring (protocol-oriented). The *x*AMp implementation consists of an integrated package, designed to be used over broadcast local-area networks. We have reasonable confidence on the utility of the several xAMp functions, thanks to the scrutiny of the several DELTA-4 consortium partners[2], who have been demanding users of xAMp.

The paper is organized as follows: section 2 provides some comparison with related approaches and the following section discusses the requirements for group support. In section 4, the architecture of *x*AMp is summarized and the protocols used to implement the different qualities of service are presented in Section 5. Section 6 presents the current state of the implementation and provides some concluding remarks.

2 Related Work

There are quite a few good algorithms published, providing individual group communication properties, like causal [27,24], total order [22], atomic [9,23,16,12], best-effort [10]. If a group communication support is to be provided, it must supply a range of functionality — let us call it *quality of service* or *QOS*, a terminology very used in the communications community — including addressing modes, group management support, and delivery properties. There exist a number of solutions providing varying degrees of order, such as [7,18,14,24]. Our subsystem, besides a range of order properties, from total and causal to FIFO, provides different agreement and synchronism properties, such as best-effort and at-least, or loose and tight synchrony[3]. Their combination, according to user requirements, yields the different qualities of service that xAMp offers.

We follow the approach, pioneered by Birman [4], of encapsulating in a group communication subsystem a set of QOSs offered to users, alleviating them from the task of constructing such functionality. The alternative approach of providing a single basic all-purpose primitive, as followed in [24], allows fine-tuning but leaves the responsibility of constructing the necessary

[1]The AMp provided only an atomic multicast *service*.

[2]Delta-4 is a consortium sponsored by the CEC Esprit II research programme, 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.

[3]The definition of these and the rationale behind their utility is detailed ahead in the paper.

services to the higher levels. We also try to keep the advantages of the latter, by providing a set of primitives, well tested (correctness) and optimized (performance), rich enough to represent most distributed application requirements.

With regard to the engineering of these multi-primitive services, these are several approaches possible. The first ISIS protocol suite [4], had two different protocols, which required a third, special protocol to enforce consistency among them. The approach taken by Malley & Peterson [21], consists of providing "micro-protocols" providing basic properties, and interconnecting them proceduraly, to obtain a given quality of service. While it is the most versatile idea, it may prove difficult to efficiently implement and combine protocols with individual properties, mainly if not only different order but also agreement and synchronism properties are envisaged. The xAMp approach consists of a core protocol, from which all combinations of properties are derived. Most services follow common execution paths, and then branch to specific terminations. With this approach, code is re-used and structures are shared, it is easier to enforce group monitoring and consistency among the information streams of the several services. In the most recent ISIS group communication subsystem, all order properties are also built on top of a basic causal protocol [7].

Looking at other protocols with more detail: Chang [9] describes an atomic broadcast protocol where requests pass through a centralized token holder. The degree of tolerance of node failures can be parametrized, in a trade-off with efficiency. A significant latency, which is not bounded a priori, may build-up due to the method to tolerate failures of the token site. The work of Navaratnam [23] is inspired on the method of Chang. Garcia-Molina gives a protocol in [14], inspired on Chang's centralized ordering node, but instead of a node there is a forest of nodes, in a graph-oriented scheme; their method does not fully take advantage of multicasting facilities and efficiency depends on groups being reasonably static. The protocol by Kaashoek [16] is equivalent to the non-fault-tolerant protocol of Chang; it owes its efficiency to this trade-off with dependability, and to the fact that it supposes a bare machine implementation, without the overheads of an operation system. Among the few works that take advantage of the properties of broadcast LANs, such as xAMp, we can cite [9,8,22,16].

3 Requirements for group support

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.

The first services required in a group support service are, naturally, the *group membership* services. A powerful support for groups should 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 should provide primitives to perform these operations. The failure of a group member should also be detected and an indication of the event should be provided to the remaining members.

The second goal of a group support service should be to provide an efficient and versatile support for exchange of information between group members. To start with, a *multicast* communication service should avoid the need to explicitly perform point-to-point transfers to execute a multicast operation. Such a service should accept a list of addresses, what we call a *selective address*, as a valid destination address for a multicast message and would – transparently – deliver the message to the intended recipients. Additionally, a *logical address* can be associated

Table 1: *x*AMp Properties.

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_{to} of the recipients, any message delivered to a recipient, is delivered to all correct recipients in P_{to}.**
- **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_{to} of the recipients, any message delivered to a recipient, is delivered to all correct recipients in P_{to}, 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.**

---

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 a group support 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 exchange 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 weaker 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 a versatile group communication service should be able to provide several qualities of service.

4 Assumptions about the xAMp architecture

The algorithms and protocols to implement the services described in the previous section strongly depend on the target architecture. The xAMp follows a low-level approach without compromising openness and portability, by using standard local area networks. LANs have architecture and technology attributes which can be used for improved performance and dependability (eg. broadcast/multicast, bounded number of omission errors, bounded transmission delay). Although designed for LANs, xAMp does not depend on a given local area network in particular. This was achieved by defining an abstract network interface, discussed with detail in [29]. We recapitulate its properties here, in table 2. Having our protocols tuned for LANs does not mean we have overlooked the problem of interconnected networks. We argue that in an interconnected networking scenario protocols can be more efficient if they rely on low-level "local" protocols that recognize important properties of the local networks. Our work has provided efficient solutions for the local scope [29], that we are now extending to interconnected networks [34].

Protocol design assumes that communication components have a fail-silent behavior. When high coverage is required, the use of self-checking components must substantiate this assumption. Tests performed in the Delta-4 project have shown however that coverage of the assumption for off-the-shelf hardware is largely acceptable for applications requiring up to a moderate level of fault-tolerance.

The xAMp architecture was designed in order to meet high expectancies with regard to fault-tolerance and real-time. The highly reliable and timely environment yielded by a single LAN used in a closed fashion had also to do with the LAN-based approach taken. We carefully devised a dependability model and established its correctness in [29], for such an environment. The basic

Table 2: Network Properties.

- **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.

protocols of our system, although clock-less (they do not require clocks), are synchronous, in the sense that known and bounded execution times are enforced, using the techniques described in [32]. Our subsystem comprises a global time service, made of approximately synchronized local clocks. Since clock synchronization is a complex issue on its own, it will not be dealt with detail in this paper. The interested reader may refer to [30], where the xAMp clock synchronization service is described in detail. With the help of this time service, a *clock-driven* protocol is built, exhibiting the tight-synchrony offered by protocols like those in [12,17]. In that sense, our system offers a more complete solution than either the asynchronous systems, or the latter synchronous systems that are only clock-driven[4].

5 Protocols: from unreliable to tight multicast

This section describes protocols which combine properties of table 1 in order to achieve a number of qualities of service. The selection of the latter 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.

5.1 The transmission with response procedure

The abstract network service, upon which xAMp relies, offers an *unreliable multicast* service, presenting a set of properties which are most useful to implement reliable multicast primitives. In absence of faults, the *broadcast* (Pn1) and *full duplex* (Pn4) properties provide message delivered to any processor connected to the network. 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 xAMp, omission errors are detected and recovered using a transmission with response procedure: it uses acknowledg-

[4]Although our clock-driven solution is not as efficient, partly because these systems use space redundancy, i.e. replicated networks, making a comparison difficult anyway.

ments to confirm the reception of the message and detects omission errors based on the *bounded omission degree* property of the abstract network[5].

Figure 1:

tr-w-resp $(\langle m\rangle, ord, send, \mathcal{M}_r, \mathcal{P}_r, n_r)$

```
// $\langle m\rangle$ is a message to be sent ($\mathcal{D}_{\langle m\rangle}$ is the set of recipients).
// "ord" is a boolean specifying if network order is relevant.
// "send" is a boolean that allows the first transmission to be skipped.
// $\mathcal{M}_r$ is a bag of responses.
// $\mathcal{P}_r$ is a set of processor from which a response is expected (usually $\mathcal{P}_r = \mathcal{D}_{\langle m\rangle}$).
// $n_r$ is the number of responses expected (usually $n_r = \#\mathcal{P}_r$).

retries := 0;
do // while
   if ($retries = 0 \vee ord$) then $\mathcal{P}_w := \mathcal{P}_r$; $n_w := n_r$; $\mathcal{M}_r := 0$; fi
   if ($retries > 0 \vee send$) then send ($\langle m\rangle$); fi
   timeout := 0; start a timer; // wait responses ...
   while ($n_w > 0 \wedge \neg timeout$) do
      when response $\langle r_m\rangle$ received from processor $p \wedge p \in \mathcal{P}_w$ do
         add $\langle r_m\rangle$ to $\mathcal{M}_r$. $n_w := n_w - 1$; remove p from $\mathcal{P}_w$. od
      when timer expires do
         timeout := 1; od
   od
   $retries := retries + 1$;
while ($retries < MAX \wedge n_w > 0$) // do
if ($n_w > 0$) then check membership fi
```

The *tr-w-resp* procedure[6] is depicted in figure 1. 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, which are then inserted in a response bag and exits when the desired number of responses is collected. If some responses are missing, the response bag is re-initialized and the message re-transmitted. The main loop finishes when all the intended responses are received or when a pre-defined retry value is reached.

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 (by setting the flag to false). Section 5.3 explains how this feature is used to provide some of xAMp 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

[5]The detailed technique, as well as its advantages over other approaches such as diffusion based masking is discussed in detail in [32].

[6]It is a modified version of the procedure given in [29].

appropriate *tr-w-resp* instantiation (also called an *emitter-machine*)[7]. 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. The unique message identification is disseminated with the message within an xAMp protocol header common to all xAMp frames. The protocol header contains the identification of the sender, the destination selective list, a frame type field and the message identification among other information[8].

5.2 Best-effort agreement

The *tr-w-resp* procedure is used in xAMp to provide reliable *frame* delivery[9]. Activated by the sender of a message, the latter must remain correct during the execution of the protocol, otherwise the number of recipients of the message cannot be determined a priori. A very efficient communication primitive is offered this way by the xAMp, under the name of *bestEffort*. From the point of view of the sender, *bestEffort* is just a call to *tr-w-resp*. The appropriate choice of $\mathcal{P}_r$ and n_r allows an early return in case of omissions, when not all the addressed recipients need to receive the message. For instance, when $n_r = 0$, the procedure immediately exits after sending the message without waiting for replies, being equivalent to *unreliable multicast*. The recipients are only required to provide an acknowledgment to the sender and to discard duplicates. The protocol is depicted in fig. 2. This primitive and the next one are helpful in a number of distributed applications where high-level functionality reduces the order and agreement requirements, but the need for efficient dissemination on group is retained.

Figure 2:

bestEffort quality of service

```
// sender
// $\mathcal{P}_r \in \mathcal{D}_{\langle m \rangle} \wedge n_r < \#\mathcal{P}_r$
when user requests to send $\langle m \rangle$ do
    tr-w-resp ($\langle m \rangle, 0, 1, \mathcal{M}_r, \mathcal{P}_r, n_r$);

    if ($\#\mathcal{M}_r \neq \#\mathcal{D}_{\langle m \rangle}$) then check membership fi
od
```

```
// receiver


when message $\langle m \rangle$ received from processor $p$ do
    send ( $\langle ok_m \rangle$ );
    if ( $\neg$ accepted$_{\langle m \rangle}$ ) then accepted$_{\langle m \rangle}$:=1; fi
od;
```

5.3 AtLeast agreement

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, what we call *atLeast* quality of service, 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 fig. 3. However, to avoid superfluous re-transmissions of the data message, recipients skip the first step of the *tr-w-resp*

[7]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.

[8]The interested reader can refer to [26]

[9]A frame is a piece of information in transit in the LAN. It may encapsulate a message or protocol control information.

procedure, using the "send" boolean parameter (see section 5.1). 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. 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.

Figure 3:

atLeast/reliable quality of service

```
// sender
// $\mathcal{P}_r \in \mathcal{D}_{\langle m \rangle} \wedge n_r < \#\mathcal{P}_r$
when user requests to send $\langle m \rangle$ do
    tr-w-resp $(\langle m \rangle, 0, 1, \mathcal{M}_r, \mathcal{D}_{\langle m \rangle}, \#\mathcal{D}_{\langle m \rangle})$;

    if $(\#\mathcal{M}_r \neq \#\mathcal{D}_{\langle m \rangle})$ then check membership fi
od
```

```
// receiver

when message $\langle m \rangle$ received from processor $p$ do
    send ( $\langle ok_m \rangle$ );
    if ( $\neg$ accepted$_{\langle m \rangle}$ ) then
        accepted$_{\langle m \rangle}$:=1;
        tr-w-resp $(\langle m \rangle, 0, 0, \mathcal{M}_r, \mathcal{D}_{\langle m \rangle}, \#\mathcal{D}_{\langle m \rangle})$;
        if $(\#\mathcal{M}_r \neq \#\mathcal{D}_{\langle m \rangle})$ then check membership fi
    fi
od
```

As with *bestEffort*, several agreement variants of *atLeast* are obtained by an appropriate choice of the $\mathcal{P}_r$ and n_r parameters. For instance, if n_r is chosen such that $n_r < \mathcal{D}_{\langle m \rangle}$, the primitive will assure that *at least* n_r of the addressed processors will receive the frame. This might be satisfactory to implement quorum based protocols. In certain passive or semi-active replication management protocols, one may wish, for performance reasons, that the message reaches all replicas, whereas is mandatory for consistency that it reaches at least the active replica. In this case $\mathcal{P}_r$ is set with the host identification of the active replica. When $\mathcal{P}_r = \mathcal{D}_{\langle m \rangle}$ (that is, when all group members do receive the message), this primitive is also called *reliable* multicast. Reliable multicast will be used as the base of two other qualities of service: *causal* and *delta*.

5.4 Causal multicast

The *reliable* quality of service does not try to impose any ordering constraints on the messages exchanged. However, in many systems the relative order of messages has a special relevance. A particular example is the FIFO order: if a given processor sends two messages there is a probability of the second message being causally related with the first one. Due to this reason, most point-to-point systems deliver messages in the order they were sent. When the interactions between participants extend across several nodes a similar reasoning can be applied. In effect, causal relations in a distributed system can be subtle and difficult to identify, specially when there are several communication paths between participants, including real world interactions (eg. sensors and actuators). For simplicity, we limit our analysis here to systems where participants only interact through message exchange using xAMp. The protocol itself restrains the sources of causal relations. Generalizing, if a processor sends a message after having received one, there is a *potential causal relation* [19] between the message sent and the message received. Several authors discussed the advantages of respecting this kind of order in a system [4,24,14]. We look for a protocol that preserves this implementation of *causal* order, which has also been called *logical order*:

> **Logical Order ($\xrightarrow{l}$):** *m2 is delivered after m1 if: 1) m2 is sent after m1 by the same processor* **or** *2) m1 is delivered to the sender of m2 before m2 being sent* **or** *3)* $m1 \xrightarrow{l} m3 \wedge m3 \xrightarrow{l} m2$.

Figure 4:

causal quality of service

```
// sender
when user requests to send $\langle m \rangle$ do
    let $h_m := \mathcal{H}_{local}$;
    xAMp ( reliable, $\langle h_m, m \rangle$ );
    add $\langle m \rangle$ to $\mathcal{H}_{local}$;
od
```

```
// receiver
when $\langle h_m, m \rangle$ message received do
    keep $\langle m \rangle$ until $h_m$ is stable; od
when $h_m$ becomes stable do
    add $h_m$ to $\mathcal{H}_{local}$; add $\langle m \rangle$ to $\mathcal{H}_{local}$;
    deliver $\langle m \rangle$; $delivered_m$:=1; od
```

In order to provide a *causal* quality of service, in addition to the assured delivery provided by the *reliable* quality of service, we need to develop a mechanism allowing logical order to be preserved. There are several implementations of logical order: using *logical clocks* [19], using message serialization [9], exchanging histories [4,24] or the more recent *vector clocks* [27,6]. In the xAMp, logical order is obtained using *causal histories*, that is, keeping a record of the messages sent and received and exchanging this information along with the data messages. A causal history is a list of *causal pairs* "$(id_{\langle m \rangle}, \mathcal{D}_{\langle m \rangle})$", where $id_{\langle m \rangle}$ is the message identifier and $\mathcal{D}_{\langle m \rangle}$ is the set of message recipients. A message sent through *causal* quality of service always carries the causal history of its sender. A causal history is updated every time a message is sent or delivered. When a message is sent, its causal pair is added to the sender's causal history. When a message is delivered, the message's pair and the associated causal history, h_m are added to the recipient's causal history (see fig 4).

In order to be delivered a message must become *stable*. A given message $\langle m \rangle$ is stable, in a given processor k, as soon as all messages in h_m have already been delivered. More precisely, and since some messages in h_m could be not addressed to k, $\langle m \rangle$ becomes stable in k when, for all precedent messages n, such that $(id_{\langle n \rangle}, \mathcal{D}_{\langle n \rangle}) \in h_m \wedge p \in \mathcal{D}_{\langle n \rangle}$, the flag $delivered_n$ is already true in k. To prevent the infinite growth of causal histories we use the synchronism of the underlying *reliable* quality of service. Let Δ be the maximum execution time for this quality of service. By definition, any message $\langle m \rangle$ becomes stable within Δ real time after being sent: thus, Δ can be used to periodically remove stable identifiers from causal histories.

Our approach is similar to that of PSYNC [24] but extended to cope with non-uniform addressing. Note that while vector clocks seem to be pretty efficient at eliminating unnecessary logical orderings sometimes enforced by the other approaches, we do not use them because our addressing scheme is too flexible to adequately support them: two consecutive messages can be sent to totally disjoint destination sets, thus a single clock is not able to represent all causal relationships. Recent implementations of ISIS [7], suggest extensions to vector clocks for several groups, but these are difficult to implement in a system as ours, where the number of different destination sets can be very large.

5.5 The atomic and tight qualities 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 media (abstract network property Pn5). However, the occurrence of omission faults, forcing the re-transmission of messages, may disturb this natural serialization. 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 *unnaccepted* 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.

Since the message cannot be delivered until the *accept* frame is received, the protocol can be further enhanced to tolerate temporarily *inaccessibility* of a recipient, that is, to allow a receiver to discard a incoming message due to a temporary lack of resources like buffer overflow. It that case the recipient should return a negative acknowledge $\langle nok_m \rangle$ to the sender. If, upon collecting all responses, the sender receives some negative acknowledgments, it issues a *reject* instead of the *accept*. Upon reception of the reject, all recipients discard the correspondent message.

The operation of the protocol is depicted in fig. 5. To save space, the extended *tight* quality of service is presented in the figure: *atomic* service can be obtained simply by removing lines 8, 11, 14 and 18. It consists of a **two-phase accept** protocol that resembles a commit protocol where the *sender* coordinates the protocol: it sends a message, implicitly *querying* about the possibility of its acceptance, to which recipients reply (dissemination phase). In the second phase (decision phase), the sender checks whether *responses* are all affirmative, in which case it issues an *accept* – or *reject*, if otherwise. To ensure the reception of the decision, by all correct recipients, the *accept* and *reject* frames are also sent using the *tr-w-resp* procedure. The two-phase accept protocol has a variant that is also depicted in fig. 5. This variant, known as the *negatively acknowledged accept*, consists in avoiding the second series of acknowledgments for improved performance. In this variant the sender transmits the accept only once and no acknowledgment is generated. If an omission affects the dissemination of the accept message, this will be recovered by the "request-decision" procedure. In the latter scenario, termination of the protocol is delayed but due to the low error rate expected in local area networks, throutput 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) according-

Figure 5:

two-phase accept (used in atomic and tight qualities of service)

```
// sender

when user request to send $\langle m\rangle$ do
   tr-w-resp $(\langle m\rangle, 1, 1, \mathcal{M}_r, \mathcal{D}_{\langle m\rangle}, \#\mathcal{D}_{\langle m\rangle})$;
   if $(\forall r \in \mathcal{M}_r, r$ is of type $\langle ok_m, \mathcal{Q}\rangle)$ then
      choose $ip_{\mathcal{Q}}$; // (Tight only)
      tr-w-resp $(\langle acc_m, ip_{\mathcal{Q}}\rangle, 0, 1, \mathcal{M}_r, \mathcal{D}_{\langle m\rangle}, \#\mathcal{D}_{\langle m\rangle})$;
      // if neg. ack: just send $\langle acc_m\rangle$ once
   else
      tr-w-resp $(\langle rej_m\rangle, 0, 1, \mathcal{M}_r, \mathcal{D}_{\langle m\rangle}, \#\mathcal{D}_{\langle m\rangle})$;
   $sent_{\langle m\rangle}$ := 1;

when receives $\langle rd_m\rangle \wedge sent_{\langle m\rangle}$ do
   send $(\langle acc_m\rangle)$;
```

```
// receiver

when message $\langle m\rangle$ received from processor $p$ do
   remove $\langle m\rangle$ from $\mathcal{Q}$;
   if ( I am accessible for $\langle m\rangle$ ) then
      add $\langle m\rangle$ to $\mathcal{Q}$; send ( $\langle ok_m, \mathcal{Q}\rangle$ ); start ( $wdTimer_{\langle m\rangle}$ );
      lock $\mathcal{Q}$; // (Tight only: no message can be consumed)
   else
      send ( $\langle nok_m\rangle$ );
   fi; od
when message $\langle acc_m, ip_{\mathcal{Q}}\rangle$ received from processor $p$ do
   stop ( $wdTimer_{\langle m\rangle}$); send ( $\langle ok_{acc}\rangle$ ); $accepted_{\langle m\rangle}$=1;
   re-order $\mathcal{Q}$; unlock $\mathcal{Q}$; od // (Tight only)
   // if neg. ack: no need to send $\langle ok_{acc}\rangle$
when message $\langle rej_m\rangle$ received from processor $p$ do
   stop ( $wdTimer_{\langle m\rangle}$); send ( $\langle ok_{rej}\rangle$ ); remove $\langle m\rangle$ from $\mathcal{Q}$;
   unlock $\mathcal{Q}$; od // (Tight only)
when $wdTimer_{\langle m\rangle}$ expires do
   tr-w-resp $(\langle rd_m\rangle, 0, 1, \mathcal{M}_r, p, 1)$;
   if ( $\langle acc_m\rangle \notin \mathcal{M}_r$ ) then monitor must be called fi od
when $\langle m\rangle$ is on top of $\mathcal{Q}$ do
   deliver $\langle m\rangle$; od
```

ly. The information required to perform a monitor action on a group is supplied by the active recipients of that group. Once activated, the action of the monitor is structured recursively in two-phased transmissions, like the normal *atomic* multicast. This solves the problem of monitor failure recovery, introduced by centralizing monitoring functions: if an active monitor fails, it is replaced by another monitor, invoked, in the same way as for a normal transmission, by a recipient that detects the failure. The action starts with an investigation phase, where information about the local contexts of group members is gathered, and ends with a decision, disseminated to those members. The decision contains the new group view, after insertion of new members, or elimination of members leaving or having failed. The recovering algorithm was described in detail in [31] and formally validated [2], so we will omit here the details of the algorithm.

For real-time applications, the major disadvantage of the *atomic* service is its inability to deal with message priorities: since incoming messages are always inserted at the end of the receive queue, a message of high priority can be affected by a possibly long queue delay until delivery. This is clearly incompatible with the real-time requirements for preemption and to respect message priorities (emphasized by the design of the real-time variant of the Delta-4 architecture, also known as XPA [3]). To avoid this problem, the two-phase accept procedure must be extended to allow the negotiation of the final position of a given message: during the dissemination phase the coordinator reads the state of the queue in all recipients, using information inserted in the acknowledgment messages. After, in the decision phase it disseminates an *insertion point* along with the decision. The method is similar to the algorithm proposed by Skeen which inspired the ABCAST protocol [4]. The final position of the message in the queue is chosen by the sender based on the information gathered during the first phase of the protocol.

To support lock-step synchronous distributed algorithms (and certain input/output activities in real-time settings) one needs messages to be delivered periodically and simultaneously to every recipient. Protocols that simulate this abstraction try to be *steady* (display a constant execution time) and *tight* (deliver a message at the same time) [33]. Besides allowing preemption, the *tight* quality of service only does a best effort to improve tightness of the protocol. Better can only be done by taking a clock-driven approach. xAMp has the *delta* quality of service, which provides a global total and causal order. The protocol is not described here due to lack of space, being described in [26]. It is based on the *reliable* quality of service to assure delivery, and on the xAMp time service, to ensure order. The protocol follows the method established by Lamport [20].

Note that this mechanism does not satisfies the requirement of short preemption latency without extra architectural support. In fact, even if an urgent message is inserted on top of the receive queue, it still has to wait for the processing of the previous message. Suppose that the application is structured as a state-machine [28]. Each message could then be associated with a state-machine command. Since commands must be processed in an atomic manner, a new command can only be processed after completion of the previous command. If a command has a very long execution time a high-priority message can be strongly delayed. This behavior can only be bypassed, if commands are splitted in several sub-commands: high-priority commands could then be inserted between two sub-commands, emulating the preemption of the long envelope command. In order to facilitate this kind of programming, xAMp possess a facility that allows the user to send several messages as a whole. These messages are seen by xAMp as a single message containing several "slots" or pieces. Slotted messages require just one execution of the protocol to be disseminated and the several slots are automatically inserted in the receive queue as individual units as soon as the envelope message is accepted.

5.6 Delta QOS

The *delta* quality of service provides total global order based on virtual synchronized clocks. It can be easily implemented using the reliable multicast service and a clock synchronization service. Before being sent, the message is timestamped with the value of the local virtual clock. Upon reception, messages are ordered by the values of their timestamps. Messages with the same timestamp are ordered using the identification of its sender (we assume that it is possible to establish an order relation between processor identifiers). To assure that timestamp order is not violated, no message can be delivered before the arrival of all messages with a smaller, or same, timestamp. That is, any message must wait a worst-case time Δ for all the potentially precedent messages. Naturally, this time Δ is given by the execution time of the reliable multicast QOS plus the maximum desynchronization between virtual clocks δ. This protocol is a variant of that of [11] but using the "tr-w-resp" procedure to avoid massive retransmission. However, our protocol can exhibit higher values for the delay Δ since acknowledgments are awaited before a message is retransmitted. On the other hand, on absence of faults, our protocol sends the data message only once over the network, thus saving network bandwidth.

5.7 Group Addressing

In xAMp, we have not imposed any restriction on the destination sets, $\mathcal{D}_{\langle m \rangle}$, of a given message $\langle m \rangle$. In fact, our protocols are quite generic and are able to accept any list of nodes as a destination set. This means that the user is able to address any sub-set of nodes in the system, listing explicitly the desired recipients. This is also called *selective addressing.*

In addition, the user is able to create *groups* to which a *logical address* is automatically assigned. When using a logical address, the user relies on the protocol to deliver the message

Figure 6:

delta QOS

```
// sender                                        // receiver
when user requests to send $\langle m \rangle$ do
    let $t_m := timenow$;
    xAMp ( reliable, $\langle t_m, m \rangle$ );            when $\langle t_m, m \rangle$ message received do
od                                                   keep $\langle m \rangle$ until $t_m + \Delta + \delta$ is reached; od
                                                 when localtime is $t_m + \Delta + \delta$ do
                                                     deliver $\langle m \rangle$; od
```

to all group members. That is, the protocol must assure that the *group view*, i.e. the list of members of a given group, is maintained and used in a correct manner. For this reason, all actions that are bound to modify this view must be performed through specialized functions. There are three operations that can be performed on a group: *join*, *leave* (voluntary departure) and *failure* (involuntary departure).

The most complex function is the one that performs the join and leave operations. The complexity comes from the use of *logical addresses*: immediately after the end of the operation, the new member must start to receive messages logically addressed to the group[10] (and an old member must stop receiving them). To avoid inconsistent uses of group views, these operations must be executed as an atomic action: in the xAMp, this is obtained making all members of the group inaccessible during the join and leave operations. In the LAN context, these operations are short lasting and bounded in duration.

On the other hand, keeping failed members in the group view does not compromise protocol correctness but may imply a performance degradation since messages will always be retransmitted up to *MAX* retries. It is then desirable to remove failed stations from group views as soon as possible. In the xAMp, failures are detected during message exchange: a sender detects the failure of a recipient during the execution of the *tr-w-resp* procedure; in *atomic* and *tight* qualities of service a recipient may detect the failure of a sender by the absence of a decision frame. Upon detection of a failure, the identity of the failed station is quickly disseminated to all nodes. A special atomic message is sent, on each group whose membership was affected by the failure, providing to the user a failure indication obeying total order.

5.8 The cost of xAMp qualities of service

The xAMp protocol provides the user with different qualities of service (QOS), as shown in table 3, ranging from unreliable multicast to *atomic* multicast. Different tradeoffs between functionality and performance are provided, assuring only a number of the properties depicted in table 1. To make these tradeoffs clear, we present a short analysis of xAMp performance. Results are summarized in tables 4. The first table presents the number of rounds required to execute the protocol for best and worst case scenarios. This table also presents the number of frames exchanged during protocol execution. These results are functions of the maximum number of faults and of the number of message recipients. The second table presents the best and worst execution time – T_e – of each QOS. These results are functions of Γ, the worst case

[10]Note that the *tr-w-resp* procedure assures recovery from omission faults relying on the number of acknowledgments collected. This means that if some group member is missing from the group view, message delivery is not assured to it.

QOS	agreement	total order.	causal	queue reord.
bestEffortN	no (best effort N)	no	FIFO	no queue
bestEffortTo	no (best effort To)	no	FIFO	no queue
atLeastN	no (assured N)	no	FIFO	no queue
atLeastTo	no (assured To)	no	FIFO	no queue
reliable	all	no	FIFO	no queue
causal	all	no	yes	no queue
atomic	all or none	yes	yes	no
tight	all or none	yes	yes	yes
delta	all or none	timestamp	timestamp	timestamp

Table 3: The xAMp multi-primitive communication service

network delay, and Tr, the time required to execute a round, that is, to send a message and collect the correspondent responses. We now deduct and discuss these values.

We start by analyzing the *tr-w-resp* procedure. This procedure executes a given number of transmission with response rounds until the message is delivered with success. In the best case the message is received by all intended recipients on the first transmission, with a worst case delay given by the network transmission delay, Γ. Note that the sender has an extra delay consisting of the time required to gather the acknowledgments, $Tr^{-}(d)$. In this case the data message is transmitted once and d responses are collected. When an omissions occur, responses are awaited until the timer expires. There is then a worst case time to execute a round, $Tr^{+}(d)$ that is roughly given by the value of this timer plus some processing time. If a worst case of k omissions occur, a delay of $kTr^{+}(d) + \Gamma$ is incurred. In the latter scenario, the data message is transmitted $k+1$ times and at most $d(k+1)$ responses are generated. Also note that since the protocol is activated at all participants, the number of crash faults does not increase the number of rounds over the minimum required to mask the network omission degree.

Note that although *bestEffort*, *atLeast* and *causal* QOSs share the same values, the processing overhead is significantly different. The procedure *tr-w-resp* is only required to be executed at the sender for the *bestEffort* QOS. On the other hand, for *atLeast* QOS, this procedure must be executed both at the sender and recipients. The causal quality of service incurres in an extra processing overhead related to the update and comparison of causal histories. Similarly, the *delta* QOS being based on the *atLeast* service, it has the same costs in terms of traffic generated. However, there is a fixed delay that must be observed before a message can be delivered, thus presenting the poorest best case performance.

The *atomic* QOS involves longer termination times since acknowledgments must be awaited before the decision is disseminated. In the best case, the data message is sent, acknowledgments are gathered and an accept is disseminated (without acknowledgments). There is one round of message exchange (generating one data message and d responses) plus one decision sent. The *tight* QOS is very similar to *atomic* except that the decision must be always acknowledged, thus involving the exchange of d responses in an extra round. In both services, the message becomes ready to be delivered as soon as the decision arrives, that is $Tr^{-}(d) + \Gamma$ after the beginning of the transfer. Worst case values for these qualities of service occur when the sender fails. These scenarios are slightly more difficult to analyze since they involve the execution of the monitor function. The results are shown in the table 4 but, for sake of brevity, the justification is omitted.

Numerical results strongly depend on the actual architecture used to support the xAMp

QOS	rounds (best)	rounds (worst)	frames (best)		frames (worst)	
			data	ctl	data	ctl
bestEffort atLeast causal delta	1	$k+1$	1	d	$k+1$	$(k+1)d$
atomic	1	$2k+f+3$	1	$d+1$	$k+1$	$(d+1)(k+f+2)$
tight	2	$k+f+2$	1	$2d+1$	$k+1$	$d(k+f+2)+(f+1)$

data: data message.
ctl: control packet.
f: crash faults. k: omission faults.
d: number of recipients.

QOS	T_e (best)	T_e (worst)
bestEffort atLeast causal	$\Gamma(sz)$	$kTr^+(d)+\Gamma(sz)$
delta	$\Delta = kTr^+(d)+\Gamma(sz)$	$\Delta = kTr^+(d)+\Gamma(sz)+\delta$
atomic	$Tr^-(d)+\Gamma(sz)$	$k[Tr^+(d)+Tr^+(1)]+T_{out}^{wd}+f[Tr^+(d)+T_{out}^{mon}]+2\Gamma(sz)$
tight	$Tr^-(d)+\Gamma(sz)$	$kTr^+(d)+T_{out}^{wd}+f[Tr^+(d)+T_{out}^{mon}]+2\Gamma(sz)$

T_e: Execution time. T_i: Inconsistency time.
$\Gamma(sz)$: Abstract Network Transmit Delay (function of message size).
$Tr(d)$: Execution time of a round (send a message and to collect d responses).
T_{out}^{wd} : wait decision timeout.
T_{out}^{mon} : timeout to activate a new monitor upon active monitor failure.

Table 4: Cost of xAMp's QOS (in frames and time)

implementation. We will focus on a port of xAMp that runs as an Unix[11] device driver. We have made our measurements on a version running on SUN Sparc-Stations 1. Figure 7 presents the results for reliable QOS and fig. 8 for atomic QOS for different message sizes and number of stations. A more complete three-dimensional plot is given in fig. 9. In the fig. 7 a single line is displayed for message indication since, as shown in table 4, the reliable QOS best-case execution time is independent of the number of recipients. As the message is indicated as soon as it is received from the network, the execution time is close to $\Gamma(sz)$ (plus some processing overhead). The confirmation is only provided when all responses are collected, being provided approximately $Tr(d)$ after the indication. Naturally, the atomic QOS is more expensive since it requires all responses to be collected and the dissemination of a decision. Whilst the confirmation is given to the user as soon as the decision is taken, being the time proportional to $\Gamma(sz)+Tr(d)$, the indication is only provided when this decision is received, that is, roughly Γ after.

6 Current State and conclusions

We have presented the xAMp, a multi-primitive group communications service. The provision of different qualities of service gives the user the possibility of choosing the compromise between performance and reliability that best fits his/her requirements. The xAMp architecture exploits the fail-silent assumption and the properties of local area networks to provide services that are highly efficient on LANS. During the design of xAMp we have traded portability over arbitrary networks by efficiency and timeliness in a local scope. xAMp cannot thus be ported to inter-connected networks. Although we are studying that problem currently, it is important to signal that as it is, xAMp is very suitable for dependable real-time applications, often based on LANs.

[11]UNIX is a Registered Trademark of AT&T.

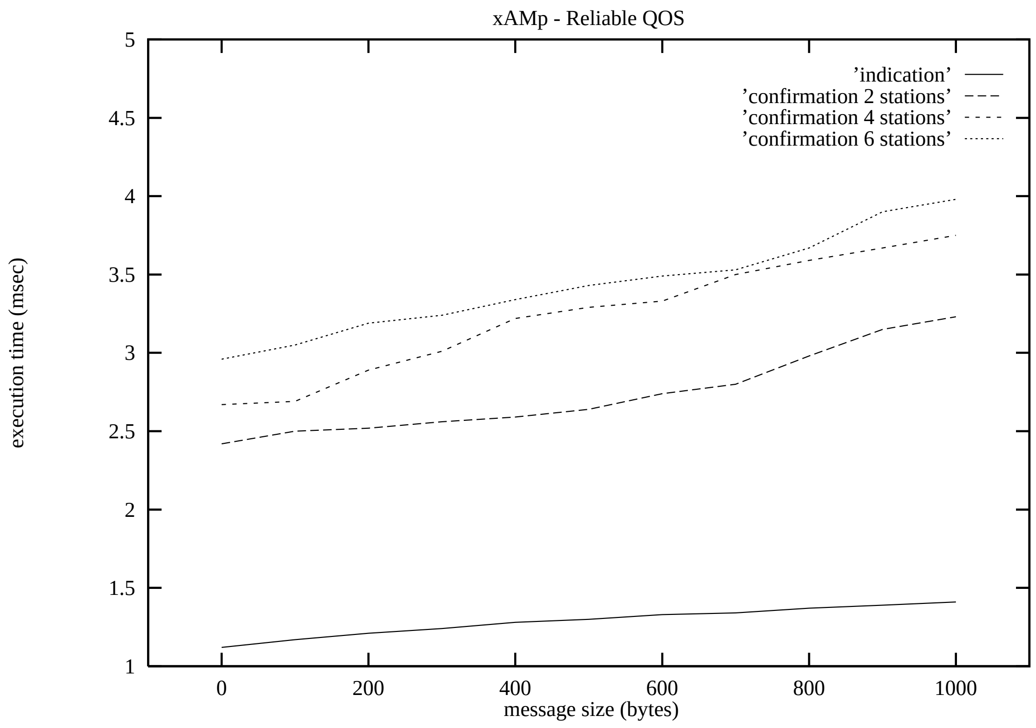


Figure 7:

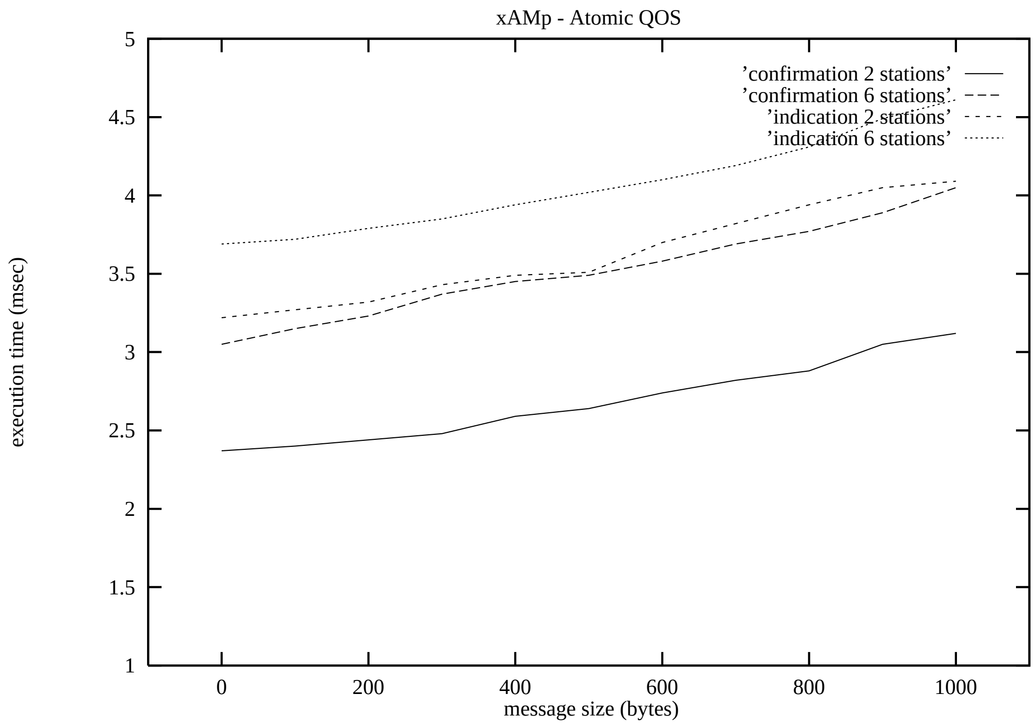
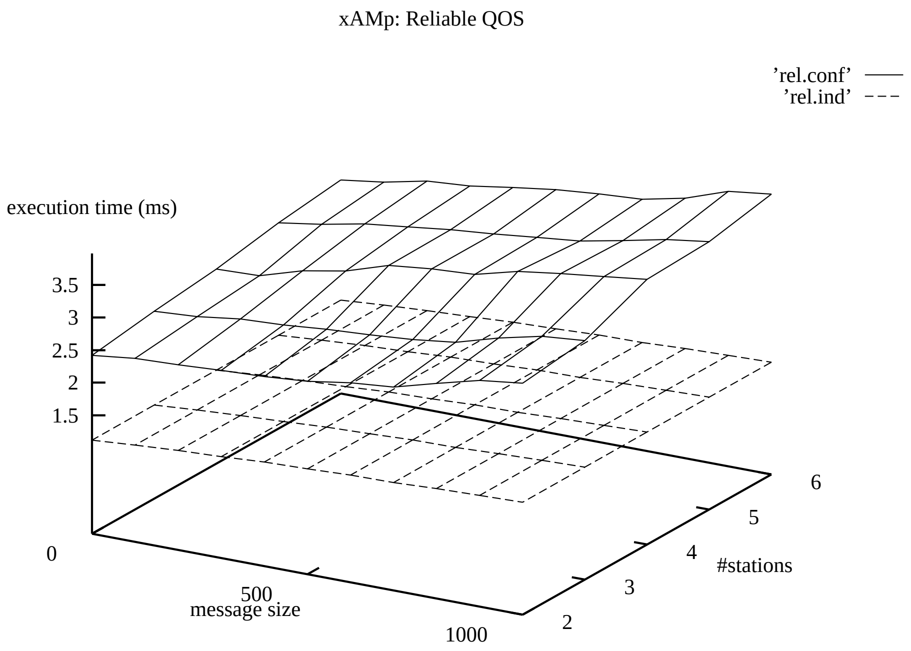


Figure 8:



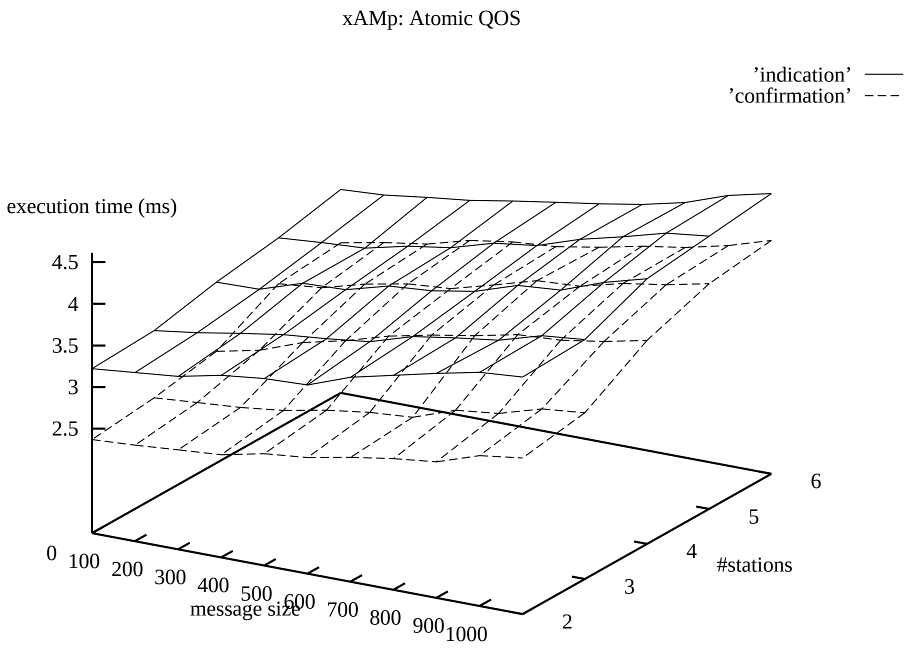


Figure 9: 3d plots

The xAMp is available as a software component consisting of a highly portable kernel and a set of interfaces to several environments and networks. Integration is a key feature of xAMp engineering. Most qualities of service are implemented as by-products of a basic core of the protocol, sharing data structures, procedures, failure-recovery algorithms and group monitor services. xAMp was easily ported to several LANs and environments, thanks to the decomposition between kernel and abstract network, and a detailed and non-ambiguous specification of all xAMp interfaces. There are currently ports to the ISO 8802 token-ring and token-bus LANs, and an FDDI port is envisaged. The performability of xAMp over each of these LANs has slight differences, depending on technology particularities. Real-time performance, net reliability and built-in fault tolerance, sheer speed and throughput, are examples of factors of choice among them. An experimental Ethernet port has also been made. Coverage of an Ethernet implementation may not be as high as over the other LANs mentioned, but it is perfectly acceptable for some non-real-time business and office segments.

The AMp specification was verified and the implementation validated. The verification tool used was Xesar [15]. The basis for the verification was an Estelle/R formal specification of the original AMp [2], forming the core of what is xAMp now. However, the verification covered most of the basic procedures, including the atomic monitor and the "tr-with-resp" procedure upon which most QOS are based, thus increasing confidence in the protocol design. The xAMp implementation was also subject to a fault injection campaign, with the help of a specialized tool [1][12].

Acknowledgments

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, J. Rufino and W. Vogels for their valuable contribution to the evolution and engineering of xAMp. 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 xAMp, particularly to P. Bond, D. Powell, JL. Richier, D. Seaton and J. Voiron. We also thank the fruitfull exchanges of ideas with Ken Birman, Larry Peterson and Rick Schlichting.

References

[1] 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*, Special Issue of Experimental C.Sc., February 1990.

[2] 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. In *Proceedings of the Third International Conference on Formal Description Techniques (FORTE 90)*, Madrid- Spain, November 1990. IFIP. Also as INESC AR/38-90.

[3] P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, D. Seaton, N. Speirs, and P. Veríssimo. The Delta-4 Extra performance architecture (XPA). In *Digest of Papers, The 20th International Symposium on Fault-Tolerant Computing*, Newcastle-UK, June 1990. IEEE. also as INESC AR/21-90.

[12] Both these works were performed collaboratively under the DELTA-4 project framework, respectively by the LGI-Grenoble and LAAS-Toulouse.

[4] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM, Transactions on Computer Systems*, 5(1), February 1987.

[5] K. Birman, T. Joseph, and F. Schmuck. Isis - a distributed programming environment, user's guide and reference manual. Technical report, The ISIS Project, Dept. of Computer Science, Cornell University, Ithaca, March 1988.

[6] Kenneth Birman, Andre Schiper, and Pat Stephenson. Fast causal multicast. Technical Report TR90-1105, Cornell University, Ithaca, USA, April 1990.

[7] Kenneth P. Birman. The process group approach to reliable distributed computing. Technical report, Cornell University, Ithaca, USA, July 1991.

[8] Michèle Cart, Jean Ferrie, and Sukrisno Mardyanto. Atomic broadcast protocol, preserving concurrency for an unreliable broadcast network. In J. Cabanel, G. Pujole, and A. Danthine, editors, *Local communication systems: LAN and PBX*. North-Holland, IFIP, 1987.

[9] J. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM, Transactions on Computer Systems*, 2(3), August 1984.

[10] D. Cheriton and W. Zwaenepoel. Distributed process groups in the V-kernel. *ACM Tran. on Computer Systems*, 3(2), May 1985.

[11] F. Cristian, Aghili. H., R. Strong, and D. Dolev. Atomic Broadcast: From simple message diffusion to Byzantine Agreement. In *Digest of Papers, The 15th International Symposium on Fault-Tolerant Computing*, Ann Arbor-USA, June 1985. IEEE.

[12] Flaviu Cristian. Synchronous atomic broadcast for redundant broadcast channels. Technical report, IBM Almaden Research Center, San Jose,California, USA, 1990.

[13] Flaviu Cristian, Robert D. Dancey, and Jon Dehn. Fault-tolerance in the advanced automation system. In *Digest of Papers, The 20th International Symposium on Fault-Tolerant Computing*, Newcastle-UK, June 1990. IEEE.

[14] H. Garcia-Molina and Annemarie Spauster. Message ordering in a multicast environment. In *Proceedings of the 9th Internacional Conference on Distributed Computing Systems*, pages 354–361. IEEE, June 1989.

[15] 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? In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, pages 275–285. Springer-Verlag, June 1989.

[16] M.Frans Kaashoek, Andrew S. Tanenbaum, Susan Flynn Hummel, and Henri E. Bal. An efficient reliable broadcast protocol. *Operating Systems Review*, 23:5–19, October 1989.

[17] Hermann Kopetz, Andreas Damm, Christian Koza, Marco Mulazzani, Wolfgang Schwabl, Christoph Senft, and Ralph Zainlinger. Distributed Fault-Tolerant Real-Time Systems: The Mars Approach. *IEEE Micro*, pages 25–41, February 1989.

[18] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Lazy Replication: Exploiting the Semantics of Distributed Services. Technical Report MIT/LCS/TR-84, MIT Laboratory for Computer Science, 1990.

[19] Leslie Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *CACM*, 7(21), July 1978.

[20] Leslie Lamport. Using Time Instead of Timeout for Fault-Tolerant Distributed Systems. *ACM Transactions on Prog. Lang. and Systems*, 6(2), April 1984.

[21] Sean W. Malley and Larry L. Peterson. A new methodology for designing network software. Technical report, University of Arizona, Tucson, USA, 1990.

[22] P.M. Melliar-Smith and L.E. Moser. Fault-tolerant distributed systems based on broadcast communication. In *Proceedings of the 9th Internacional Conference on Distributed Computing systems*, pages 129–133. IEEE, June 1989.

[23] S. Navaratnam, S. Chanson, and G. Neufeld. Reliable group communication in distributed systems. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 428–437, San Jose - USA, June 1988. IEEE.

[24] 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.

[25] D. Powell, D. Seaton, G. Bonn, P. Veríssimo, and F. Waeselynk. The Delta-4 approach to dependability in open distributed computing systems. In *Digest of Papers, The 18th International Symposium on Fault-Tolerant Computing*, Tokyo - Japan, June 1988. IEEE.

[26] L. Rodrigues and P. Veríssimo. xAMp, a protocol suite for group communication. Technical Report RT/43-92, INESC, Lisboa, Portugal, January 1992.

[27] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *Proceedings of the 3rd Int Workshop on Distributed Algorithms*, volume LNCS 392, pages 219–232, Nice - France, September 1989. Springer Verlag.

[28] F. B. Schneider. The state machine approach: a tutorial. In *Proceedings of the Workshop on Fault-tolerant Distributed Computing*, Lecture Notes in Computer Science. Springer-Verlag, 1988.

[29] P. Veríssimo and José A. Marques. Reliable broadcast for fault-tolerance on local computer networks. In *Proceedings of the Ninth Symposium on Reliable Distributed Systems*, Huntsville, Alabama-USA, October 1990. IEEE. Also as INESC AR/24-90.

[30] 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*, Boston - USA, July 1992. INESC AR/65-92.

[31] P. Veríssimo, L. Rodrigues, and M. Baptista. AMp: A highly parallel atomic multicast protocol. In *Proceedings of the SIGCOM'89 Symposium*, Austin-USA, September 1989. ACM.

[32] P. Veríssimo, J. Rufino, and L. Rodrigues. Enforcing real-time behaviour of LAN-based protocols. In *Proceedings of the 10th IFAC Workshop on Distributed Computer Control Systems*, Semmering, Austria, September 1991. IFAC.

[33] Paulo Veríssimo. Real-time data management with clock-less reliable broadcast protocols. In *Proceedings of the Workshop on the Management of Replicated Data*, Houston, Texas-USA, November 1990. IEEE. also as INESC AR/25-90.

[34] Werner Vogels and Paulo Veríssimo. Process groups and reliable multicast communication in extended lans, mans and internetworks. Technical report, INESC, August 1991.