Semantically Reliable Multicast: Definition, Implementation and Performance Evaluation*

José PEREIRA
Universidade do Minho
jop@di.uminho.pt

Luís RODRIGUES
Universidade de Lisboa
ler@di.fc.ul.pt

Rui OLIVEIRA
Universidade do Minho
rco@di.uminho.pt

Abstract

Semantic Reliability is a novel correctness criterion for multicast protocols based on the concept of message obsolescence: A message becomes obsolete when its content or purpose is superseded by a subsequent message. By exploiting obsolescence, a reliable multicast protocol may drop irrelevant messages to find additional buffer space for new messages. This makes the multicast protocol more resilient to transient performance perturbations of group members, thus improving throughput stability.

This paper describes our experience in developing a suite of semantically reliable protocols. It summarizes the motivation, definition and algorithmic issues and presents performance figures obtained with a running implementation. The data obtained experimentally is compared with analytic and simulation models. This comparison allows us to confirm the validity of these models and the usefulness of the approach. Finally, the paper reports the application of our prototype to distributed multi-player games.

KEYWORDS: reliable multicast protocols; throughput stability; semantic reliability.

*Partially funded by FCT SHIFT project (POSI/32869/CHS/2000).

1 Introduction

In the context of reliable distributed computing one is often faced with the need to provide services that offer not only strong consistency guarantees but also good performance. Conciliating these two goals is usually a difficult task. When considering multicast communication, strong reliability can be expressed informally in the following way: If a message m is delivered to a correct process, m is delivered to every correct process. To enforce this property, a reliable multicast protocol is required to store messages until they are *stable*, *i.e.*, until their reception has been acknowledged by all group members. Under high loads, a single slow member may prevent messages from becoming stable at the same pace they are being produced. This quickly leads to buffer space shortages and to global performance degradation due to flow control, limiting the applicability of reliable multicast [28].

This problem can be circumvented by relaxing the reliability of multicast, for instance, by not delivering all messages to perturbed processes [5]. Unfortunately, when strong reliability is lost, most of the simplicity that was gained at the application level is also lost. We have recently proposed an alternative approach to tackle this problem using a novel correctness criterion called Semantic Reliability [25, 26, 27]. The proposed model is based on the concept of *message obsolescence*: A message becomes obsolete when its content or purpose is superseded by another message. This knowledge is used by a semantically reliable protocol to selectively discard some messages from buffers in the presence of overload conditions. By allowing obsolete messages to be discarded, the system tolerates better the occurrence of performance perturbations without demanding the allocation of additional resources. Still, slow processes are guaranteed to receive all the non-obsolete information, thus approaching the convenience of full reliability.

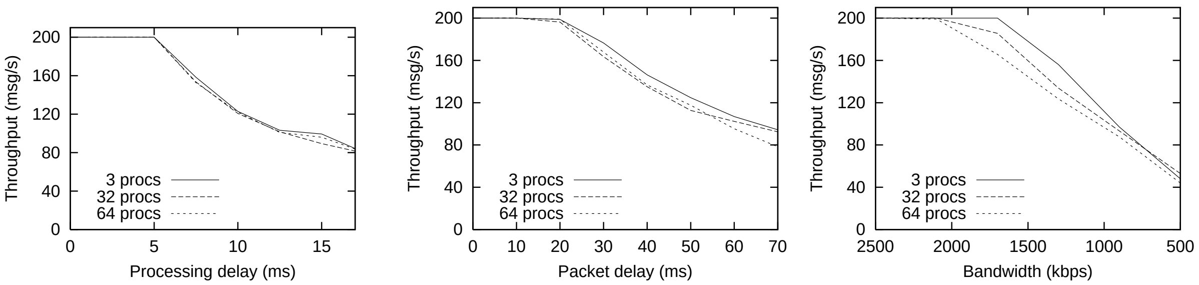
The set of protocols that we have designed includes Semantically Sender-Reliable Multicast (S-SM) [25], Semantically Reliable Multicast (S-RM) [26], and Semantically View Synchronous Multicast (S-VSM) [27], combined with ordering guarantees [24]. The current paper summarizes the motivation and definition of semantically reliable protocols, as well as the algorithmic issues arising in their implementation. In addition, it makes the following contributions: *i)* It describes a prototype implementation of the protocols and presents the performance measurements obtained with it. The data collected experimentally is compared with analytic and simulation models, confirming their validity. *ii)* The prototype implementation allows also to draw conclusions on the overhead and scalability of protocol mechanisms required for semantic reliability. *iii)* Finally, the paper reports the application of our prototype to distributed multi-player games.

The paper is structured as follows: In Section 2 we address the issue of multicast flow control and its role in the performance of heterogeneous multicast groups. Section 3 introduces the concept of message obsolescence and shows how it can be expressed by the application at the protocol interface. Section 4 summarizes specifications and algorithms for semantically reliable protocols. Sections 5 describes analytical and simulation models as well as the prototype implementation. Section 6 compares performance results and discusses system configuration issues. Section 7 illustrates the protocol using a concrete application. Section 8 compares our proposal with related work and Section 9 concludes the paper.

2 Motivation

The problem of achieving and sustaining high multicast throughput is intrinsically related to flow control in multicast protocols. A multicast system, composed of a sender, intermediate network (links and routers), and receivers, can be described as a pipeline. Each stage of the pipeline has a maximum capacity, determined by characteristics such as processing power, memory or bandwidth. If input continually exceeds the capacity of any given stage, that stage becomes overloaded and its performance degrades, affecting the entire message flow. For instance, when overloaded, a network can exhibit a much lower bandwidth than its maximum capacity [17].

Flow control mechanisms in network protocols ensure that the source does not produce more messages than any recipient or network component can handle, thus enabling full but safe use of available resources. This is commonly achieved by dynamically evaluating resource availability and adapting to it, namely, using the classical window-based mechanism as in TCP/IP [17]. Individual stages of the pipeline tolerate transient performance perturbations



(a) Processing delay upon delivery

(b) Packet delay in the network

(c) Network link bandwidth

Figure 1: Throughput degradation of reliable multicast when a single receiver is perturbed.

by temporarily buffering messages. When storage space becomes exhausted they propagate this information to the previous stage. Eventually, the source is reached and forced to diminish its sending rate. Back-pressure on the preceding stage can be established through explicit messages or implicitly, for instance by not acknowledging the reception of previous messages.

In the context of multicast communication all recipients and links to recipients are part of a common pipeline rooted at the multicast source. Regardless of the specific flow control mechanism used, a single slow recipient eventually forces the source to slow down, degrading overall group performance. This behavior can be illustrated by perturbing a single element of a multicast group and observing the throughput achieved by other members.

Figure 1 shows the average throughput in messages per second (y axis) for increasingly greater perturbations introduced to one of the receivers (x axis) as measured by an unperturbed receiver. We use another unperturbed element of the group as the sender, producing messages at most at a constant rate of 200 msg/s. Since at this point we are merely trying to motivate our work, a detailed description of the experimental conditions is postponed to Sections 5 and 6.

Figure 1(a) shows the effect of introducing delays upon delivery of each message. Figure 1(b) illustrates the consequences of delaying network packets. In Figure 1(c) the network bandwidth of the perturbed receiver is reduced and the message payload is incresed from 4 to 1000 bytes. Notice that for each kind of perturbation, there is a point beyond which the perturbation is too large for the receiver to to keep up with the sender, thereby forcing it to wait and affecting all other receivers as well. In addition, when the system is congested buffers are full. Transient performance degradation conditions within a single stage of the pipeline will immediately affect the whole system. Namely, the variability of the interval between messages grows because it becomes dependent on the retransmission and stability detection mechanisms.

Naturally, if reliability is strictly required, *i.e.*, if all recipients must eventually deliver all messages, either the sender adjusts to the slowest receiver or messages indefinitely accumulate for delivery within the system. Another solution would be to exclude the slowest process from the group. Unfortunately, transient problems by different machines may induce the same behavior as a consistently slow single node [28, 4].

An alternative path to address the problem is to weaken reliability requirements, so that slower receivers are not required to deliver all messages and thus do not need to slow down the sender. However, pure unreliable protocols, that randomly drop messages, are of little use to many applications. Even if some mechanism is implemented to notify the receiver that some messages have been dropped [9], the application might be unable to take any corrective measure since it has no knowledge of that message's content.

It has been proposed the parallel use of two multicast protocols: An unreliable protocol used for payload and a reliable protocol used to convey meta-data describing the content of data messages sent on the payload channel [30]. Using this information, the receiver may evaluate the relevance of lost messages and explicitly request retransmission when needed. Our approach is inspired on this principle, but exploits the semantic knowledge at the sender side instead. As we will explain later in the text, this allows us to make the same optimizations without requiring the maintenance of two parallel communication channels and without requiring the involvement of the application in managing retransmissions.

3 Message Obsolescence

The basic idea behind our approach is that in a distributed application some messages overwrite or implicitly convey the content of other messages sent in the past, therefore making them irrelevant. If a message that has not yet been delivered to the application is recognized as obsolete after the arrival of a more recent message, it can be safely purged without compromising the application's correctness. Notice that when the system is congested this is likely to happen as buffers in the path to the slower component will be full and thus contain messages sent some time ago. Over time, if enough messages can be purged, back-pressure is avoided thus sustaining a high throughput. If not, purging some messages at least ensures that back-pressure is weaker, minimizing upstream congestion.

The resulting reliability model is Semantic Reliability, as all the information is eventually delivered in spite of not delivering all messages. In order to use this concept we must identify which applications exhibit message obsolescence and show that it is possible to express this property in a generic form. In the remainder of this section we will address these two issues.

3.1 Applications with Message Obsolescence

Applications embodying operations with overwrite semantics, in particular, applications managing read-write items are the most obvious example of applications that exhibit message obsolescence. In these applications, any update of a given item is made obsolete by subsequent update operations. Recognizing this fact, some applications deal with obsolescence directly. For instance, distributed file-systems, such as NFS, cache write operations in the client to minimize network traffic [33]. Other examples include weakly consistent distributed shared memory systems, where a sequence of memory operations is bounded by synchronization primitives to delay distribution of updates [31].

However, it is not possible to implement these optimizations at the application level when using multicast protocols. In order to timely update faster receivers, the application should forward updates to the protocol as soon as possible. At that point, the message becomes out of reach of the application and cannot be discarded even if it becomes obsolete shortly afterwards. Slower receivers are thus forced to deliver already obsolete messages. Typical examples are applications such as on-line trading systems, where new quotes have to be continuously disseminated to a large number of recipients [25] or distributed multi-player games, in which frequently updated game state is disseminated to a group of replicas (see Section 7).

Not only applications with read-write semantics exhibit the obsolescence property. For instance, many distributed algorithms are structured in logical rounds and, when the algorithm advances to the next round, messages from previous rounds become obsolete. Recognizing this property, the notion of *stubborn channel* in which reliability has to be ensured just for the last messages has been proposed [13]. Notice that the number of rounds is not bounded. It has been shown how the fundamental problem of distributed consensus can be solved in asynchronous distributed systems augmented with failure detectors and stubborn channels [13, 22]. A stubborn channel can be seen as a particular case of obsolescence in which each message obsoletes all predecessors.

3.2 Expressing Obsolescence

In order to be useful for a wide range of applications, obsolescence must be expressed at the protocol interface in a generic way. Furthermore, the interface must not be tied to message content, to ensure that protocol and application implementations can be kept separate. We are interested in general purpose techniques that can be applied to a wide range of systems in an efficient manner. Therefore, we exclude solutions such as requiring special formatting of messages [7] or enriching the messages with code [36].

All the protocol requires to determine obsolescence is a binary relation on messages. The *obsolescence relation* is a strict partial order (*i.e.* anti-symmetric and transitive) and a subset of causal ordering of events. In this text, the fact that a message m is *obsoleted* by a message m' is expressed as $m \sqsubset m'$. The intuitive meaning of this relation is that if $m \sqsubset m'$ and m' is delivered, the correctness of the application is not affected by omitting the delivery of m. The notation $m \sqsubseteq m'$ is used as a shorthand for $m \sqsubset m' \vee m = m'$.

The obsolescence relation can then be encoded by the application and be conveyed to the protocol. Upon multicast of a message m, the protocol is informed, using an appropriate data structure, of messages that become obsolete. The representation technique must be compact, as annotations need to be stored in buffers and transmitted over the network. Representation should also favor time and space efficient algorithms and data structures to manipulate protocol buffers and determine obsolete messages. Finally the encoding should be able to represent a large subset of possible obsolescence relations. This is somewhat simplified by the representation not needing to be complete to ensure application correctness: even if some messages are not recognized as obsolete, only purging efficiency is lost.

The technique proposed here exploits the fact that purging is mainly applied to pairs of messages that are close to each other in the message stream and works as follows. Each message explicitly enumerates which of the k preceding messages multicast by the same process it makes obsolete. This information can be stored in a bitmap of k bits. If the n^{th} position of the bitmap is set to *true*, the message makes obsolete the n^{th} preceding message. This is not only extremely compact but also makes it very easy to compute the representation of transitive obsolescence relations using only shift and binary "or" operators. It also makes it very easy to compute, using the same efficient operators, the representation of the obsolescence relation when a message makes several other obsolete.

We have used this representation in the experimental evaluation performed in later sections of this paper and show how to select parameter k. Other techniques for representing obsolescence are discussed in [27]. The proposed technique can also be applied to represent obsolescence relations among totally ordered messages from different senders, using the bitmap to identify previous delivered messages which have become obsolete [24].

4 Semantically Reliable Multicast Services

In this section we present the specification of two representative semantically reliable multicast protocols: Semantically Sender-Reliable Multicast (S-SM) and Semantically Reliable Multicast (S-RM). The first is a simple protocol, in which purging can be implemented with minimal additional effort, while the second illustrates the main issues that emerge when implementing a semantically reliable protocol suitable for fault tolerant programming.

4.1 System Model

The specification is presented in the context of an asynchronous message passing system [15]. Briefly, the distributed system is modeled as a set of sequential processes. In each step a process can: send a message; receive a message; perform local computation; and crash. A distributed computation is characterized by the set of the steps of all processes ordered by the causality relation [20]. No assumptions are made on relative execution speed of processes or on the existence of synchronized clocks. Processes can only fail by crashing and do not recover, thus excluding byzantine faults. A process that does not crash is correct. We assume that at most f processes can crash.

Processes are fully connected by an asynchronous network of point-to-point message passing channels. Asynchrony means that there is no bound on the time that a message takes to be transmitted. A channel connecting process p to process q is used through primitives *send*(m, q) and *receive*(m, p). Briefly, reliability means that if both the sender and receiver processes are correct, the message is eventually received. Additionally we assume that channels are FIFO ordered. Assumptions on reliability and FIFO order are actually not strictly required, but used to simplify the presentation of the algorithms. In fact, channels can be reduced to fair-lossy channels [3].

The multicast service is used through primitives *multicast*(m) and *deliver*(m) [15]. If during a computation a process executes *multicast*(m) (resp. *deliver*(m)) it is said to "multicast message m" (resp. "deliver message m"). The obsolescence relation is used as described in Section 3.2.

4.2 Definition

Semantically Sender-Reliable Multicast (S-SM), a simple protocol that does not enforce consistency upon failure of a sender [25], is defined in Figure 2. Both protocols described in this section enfore FIFO order, mainly to illustrate how to combine semantic reliability and purging with order constraints. The intuitive notion that a message can be substituted by another that makes it obsolete is captured in the previous definition by the statement "deliver some m' such that $m \sqsubseteq m'$". If a message m never becomes obsolete, the protocol is required to deliver m itself. Therefore, if the obsolescence relation is empty the protocol defaults to conventional reliability.

On the other hand, if there is an infinite sequence of messages $m_1, m_2, \ldots$ such that $m_i \sqsubset m_{i+1}$ the protocol may omit all these messages, as captured by the statement "there is a time

Sender Reliability: If a correct process multicasts a message m and there is a time after which no process multicasts some m'' such that $m \sqsubset m''$, then all correct processes deliver some m' such that $m \sqsubseteq m'$.

Integrity: For every message m, every process delivers m at most once and only if m was previously multicast by some process.

FIFO Order: If a process multicasts a message m before it multicasts a message m', no process delivers m after delivering m'.

Figure 2: Definition of Semantically Sender-Reliable Multicast (S-SM).

after which no process multicasts m'' such that $m \sqsubset m''$". It may seem awkward at first that such occurrence is allowed. However, it should be noted that the application, by a judicious definition of the obsolescence relation, can easily prevent such sequences from occurring. Actually, the application can decide exactly which is the most appropriate length of any sequence of messages related by obsolescence. If the protocol was forced to deliver messages from a possibly infinite sequence (by omitting the statement above from the definition), the protocol designer would be forced to make an arbitrary decision of which messages to choose from that infinite sequence (*e.g.*, one out of every k messages). It is clearly preferable to leave this decision to the application.

Notice that the Sender Reliability property holds only when the sender of a message is correct. When a sender crashes, S-SM is allowed to deliver messages to some but not all destinations even if not obsolete, making it unsuitable for fault-tolerant programming. Nevertheless, it makes possible to have simple implementations in which each process is required to buffer and retransmit just its own messages. Besides illustrating the concept of semantic reliability, such protocol is useful as a building block [27] or for information dissemination applications with less stringent consistency requirements [25].

Semantically Reliable Multicast (S-RM) provides the required consistency guarantees even when a sender fails can be used for replication in fault-tolerant systems [26] and is defined in Figure 3. Agreement ensures that even when the sender crashes, all receivers deliver the same

Validity: If a correct process multicasts a message m and there is a time after which no process multicasts some m'' such that $m \sqsubset m''$, then it delivers some m' such that $m \sqsubseteq m'$.

Agreement: If a correct process delivers a message m and there is a time after which no process multicasts m'' such that $m \sqsubset m''$, then all correct processes deliver some m' such that $m \sqsubseteq m'$.

Integrity and FIFO Order: (as shown in Figure 2 for S-SM)

FIFO Completeness: If a process multicasts a message m before it multicasts a message m' and there is a time after which no process multicasts m''' such that $m \sqsubset m'''$, no correct process delivers m' without delivering some m'' such that $m \sqsubseteq m''$.

Figure 3: Definition of Semantically Reliable Multicast (S-RM).

set of non-obsolete messages. In addition, FIFO Completeness ensures that a replacement for all obsolete predecessors of some message m are guaranteed to be delivered for m to be also delivered.

The usefulness of S-RM for fault-tolerant replication can be illustrated by a simple example. Consider a replicated server using the primary-backup approach [6] managing a set of data items in which S-RM is used to update replicas. Each update message carries the value of an item and obsoletes previous messages regarding the same item. If the primary fails, the Agreement property ensures that the same set of non-obsolete updates is delivered by all backup replicas which therefore have the same state. FIFO Completeness ensures that only obsolete predecessors of the last message delivered have been omitted, and therefore that the state of backup replicas is the same as the state of the primary at some point in time [26, 27].

4.3 Algorithms

The correctness of an implementation of S-SM is straightforward: When a message is multicast, it is sent to each of the destinations. Both the sender and the receivers can purge obsolete messages found in local buffers. This makes it easy reuse mechanisms that exist to improve the

```
Initially:
    for all p: to-send[p] = empty queue
    for all p: received[p] = $\emptyset$
    to-deliver = empty queue

func new(Message m) =
    $\forall$ p: m $\notin$ received[p]

func safe(Message m) =
    |p: m' $\in$ received[p]$\wedge$ m $\sqsubseteq$ m'| $>$ f

proc purge_r(Q) do
    while $\exists$ m, m' $\in$ Q: m $\sqsubset$ m' $\wedge$ safe(m') do
        remove(Q, m)

proc purge_d(Q) do
    while $\exists$ m, m' $\in$ Q: m $\sqsubset$ m' do
        remove(Q, m)

$t1$ : upon multicast(m) do
    addToTail(to-send[self], $m$)

$t2$ : upon to-send[p]$\neq \emptyset$ do
    m $\leftarrow$removeFirst(to-send[p], m)
    send(m,p)

$t3$ : upon receive(m,q) do
    if new(m) then
        forall p $\in$ destinations: p $\neq$ self do
            addToTail(to-send[p], m)
        addToTail(to-deliver, m)
        purge_d(to-deliver)
    addTo(received[p], m)
    addTo(received[self], m)
    forall p $\in$ destinations do
        purge_r(to-send[p])

$t4$ : upon to-deliver $\neq \emptyset$ do
    m $\leftarrow$removeFirst(to-deliver, m)
    addToTail(delivered, m)
    deliver(m)
```

Figure 4: Algorithm for Semantically Reliable Multicast (S-RM).

performance and scalability of multicast protocols [12].

An algorithm for S-RM is outlined in Figure 4. The notation used is as follows: Each **upon/do** clause is assumed to be executed atomically. When several clauses are enabled, *i.e.*, their pre-condition is true, one of them is chosen non-deterministically to be executed (fairness assumptions are presented in the text). Sets and queues are used as auxiliary data structures. The usual notation is used for sets in addition to procedure *addTo*(S, e), that inserts an element e in the set variable S. Queues are used with procedures *addToTail*(Q, e), that inserts element e in the queue variable Q, and *remove*(Q, e), that removes element e from Q. Procedure *removeFirst*(Q) removes and returns the first element of Q.

As in reliable multicast, ensuring Agreement requires that all processes are able to relay received messages [15]. Each destination keeps track of messages already received in order not to deliver them more than once. The protocol has to take additional precautions not to violate FIFO Completeness when purging obsolete messages. Consider the following scenario and sequence of events: A process p multicasts messages m_1, m_2, m_3 such that only $m_1 \sqsubset m_3$; p purges m_1 due to m_3; m_2 is transmitted from p to some process q; q delivers m; p

crashes. Clearly, this sequence violates FIFO Completeness. The problem is that m_1 was purged before ensuring the eventual delivery of m_3. A message is guaranteed to be eventually delivered as soon as it has been received by $f + 1$ processes, where f is the maximum number of processes that may fail. When this condition holds, we say that the message is *safe*. In the particular sequence above, violation of FIFO Completeness could be avoided if purging of m_1 was delayed until m_3 was known to be safe.

Both these issues are addressed by keeping track of which messages have been received from each process in *received*$[p]$ and captured in functions *new*(m) and *safe*(m) in the algorithm presented in Figure 4. Notice that purging of the delivery queue does not need to be restricted to safe messages and thus different procedures are used to purge retransmission and delivery queues.

In detail, the algorithm works as follows: Upon multicast ($t1$) the message is queued and ($t2$) later sent only to the sender process itself. The message is then relayed to all destinations (except to the sender itself) and queued for delivery upon reception for the first time ($t3$). Messages that are never purged are eventually delivered ($t4$). This implies that transitions $t2$ and $t4$ are weakly fair, *i.e.*, cannot be enabled forever in a correct process without being eventually executed [21]. Notice also that transition $t3$ is also weakly fair, according to the assumption of reliable channels. No fairness assumption is required on $t1$, as an application that never multicasts messages is still correct. Purging of the delivery queue is performed upon reception ($t3$) only if a message has been queued. The retransmission queue is purged regardless of no new message having been queued for retransmission, as an existing message might have become safe and thus made purging possible. Notice that there is no point in purging *to-send* in $t1$, as the new message is guaranteed not to be safe yet as it is still waiting to be sent to the sender itself.

4.4 Order and View Synchrony

In this text we have shown the application of semantic reliability to a FIFO ordered reliable multicast protocol. We have also studied other orders as well as view synchrony [24]. In this section, we only address briefly some selected aspects related to the provision of these additional services.

Causal Order [15] requires the definition of Causal Completeness, similar to FIFO Completeness but considering all causal predecessors of a message instead of only those multicast by the same process. An implementation of S-RM with FIFO can easily be extended to address this [24]. The definition of View Synchrony has to be modified to consider obsolete messages [27]. Nevertheless, modifying an implementation of view synchrony for semantic reliability is straightforward, as purging can be performed independently by each process. On the other hand, Total Order as defined for reliable multicast is directly useful with semantic reliability. Modifying a protocol based on a consensus protocol [8] is also feasible: Purging can be done both on messages waiting to be proposed as well whose order has already been decided [24].

5 Implementation versus Analytical and Simulation Models

In order to assess the performance of semantically reliable multicast protocols we are interested in using three complementary techniques, namely:

A simple analytical model. The purpose of this model is to provide to application designer a simple method to configure some protocol parameters, such as buffer sizes and to assess the expected purging rate given a concrete obsolescence pattern.

A detailed simulation model. The purpose of this model is to estimate the performance of the protocol in complex settings before it is actually implemented.

Collection of experimental data from the implementation. This method allows us to measure the performance of the protocol in a real system.

Naturally, we would like to have the analytical model as simple as possible but still close to reality. Similarly, we are interested in having a simulation model that closely approximates the real implementation, such that performance estimates are accurate. Therefore, it is interesting to compare the output of both techniques, in particular to verify if there are some parameters that are not taken into consideration in the simulation model but affect the performance in the real implementation.

5.1 Analytical Model

To derive our analytical model we consider a simplified system model constituted by a single sender, a fast receiver and a slow receiver. The sender produces messages at rate T_s. For each receiver, messages are placed in a buffer with capacity for N messages. If a message cannot be inserted in one of the buffers, the sender blocks until buffer space becomes available. A fast receiver removes messages from its buffer as soon as they become available. On the other hand, the slow receiver removes messages from its buffer at rate T_r. Considering $T_r < T_s$, the slow receiver's buffer eventually fills up. When this happens, the protocol searches the buffer for obsolete messages, freeing space to store arriving messages. If the system remains overloaded for a long period, the buffer will eventually be filled just with unrelated messages. Therefore, new messages can only be accepted if they obsolete one of the messages in the buffer.

The estimation of performance thus depends on knowing the distance in the input stream between related messages. Unless obsolescence is strictly periodic, this is a random variable. Let D be the distance between each message and the latest message obsoleted by it, and $f(x) = P(D = x)$ the probability mass function of D. Value $f(0)$ is assumed to be the probability of not existing any obsoleted predecessor message.

The probability of a message being obsoleted by a new message is thus given by $R_* = \sum_{x \geq 1} f(x)$, which is an estimate of maximum ratio of messages that can be purged by the protocol under continued congestion. However, this is not a good estimate of how the protocol would behave, as it implicitly assumes an unbounded amount of previous buffered messages.

Knowing that when the system is congested buffers are full, a more reasonable assumption is to consider that buffer size determines the maximum distance between two related messages such that one of them can be discarded. Making the simplifying assumption that the buffer holds messages sent immediately before, total probability of an obsoleted predecessor existing in the buffer is thus $R = \sum_{1 \leq x \leq N} f(x)$, where N is the maximum number of messages buffered for each receiver. This gives an estimate of the ratio of messages that can be purged by the protocol under continued congestion. Using R and given maximum sender and receiver throughputs T_s and T_r, it is possible to derive the effective throughputs T (departing from the sender and being consumed by a fast receiver) and T' (being consumed by the slower receiver):

$$T = \min(T_s, \frac{T_r}{1 - R}) \tag{1}$$

$$T' = \min(T, T_r) \tag{2}$$

Naturally, if probability accumulates at low values of distance, *i.e.*, if the probability of a message being made obsolete by a close subsequent message is high, the purging procedure is very effective. On the other hand, if the distance is large, it is likely that the buffers become exhausted before any message has the chance to become obsolete. It is also clear that, for the same obsolescence distribution, the algorithm performs best for larger buffer sizes.

5.2 Simulation Model

The analytical model does not take into consideration several issues that may affect the efficiency of the algorithm. To start with, it does not consider the effect of the purging procedure itself in the content of the buffer, which means that even if exactly N messages are stored, they are likely not to be the last N messages. Thus the buffer might hold any N previous messages or even some posterior messages. Additionally, in a real system we have two buffers, one at the sender and the other at the recipient, where purging may be applied. If obsolete messages are purged in the sender's buffer, there is the possibility that some obsolete information never reaches recipients. On the other hand, there is less load imposed downstream. By using a simulation model, we can confirm the validity of the analytical model despite its simplicity and to explore the impact of system parameters in performance of purging in more complex models.

A discrete-event simulation model [18] works by keeping a queue of events ordered by their scheduled time. The simulation progresses by removing and executing the event in the head of the queue. The scheduled time of the event is considered the current time during the execution. Executing an event may change the state of the model and schedule further events to a posterior time. Simulation terminates when the event queue is empty or the time reaches a pre-established maximum.

The system state is composed by a pair of FIFO buffers with configurable size N, one for the fast and one for the slow receiver. Events are periodically scheduled to produce and consume messages according to at most T_s and T_r. The obsolescence relation can either be generated according to a random distribution or replay a profile obtained from a real application. The obsolescence relation is represented using a bitmap. The simulation logs the time when each message is produced, enters each buffer, leaves each buffer, and is consumed. Statistics are then computed from the logs.

5.3 Prototype Implementation

The prototype is implemented in C++ and uses the ACE framework [34] as an operating system abstraction layer. The protocol code is event-driven and executes in a single thread. Events can be triggered by arrival of messages, by timeouts or explicitly queued by event-handlers. Buffering by processes other than the sender and purging can be conditionally compiled, thus obtaining S-SM, S-RM and corresponding reliable multicast protocols for comparison.

Upon multicast, a message is immediately optimistically disseminated using IP multicast and then buffered. An optional upper bound on the bandwidth consumed by multicast can be imposed to avoid congesting the network. Upon reception, messages are queued for delivery. Reliability is ensured by a receiver initiated mechanism [29]. Each receiver keeps a queue of messages discovered to be missing which it requests using negative acknowledgment messages. Retransmission requests are controlled by a variable window which uses the TCP/IP algorithm [17].

Garbage collection of buffers is based on a scalable stability detection algorithm [14]. This algorithm uses gossiping to determine which is the last message that has been received by all processes. Intermediate control messages of this protocol are used by receivers as hints in the discovery that the last message sent by some process has been lost.

Flow control is performed by imposing a limit on the total number of messages buffered by the protocol (*i.e.*, for retransmission and for delivery) at each process. Therefore, if a process is consuming messages slower than they arrive, messages accumulate in its buffers. Eventually this process ceases to accept further data messages from the network which makes the stability detection algorithm block in the latest message stored by such process. Being unable to remove messages from its buffers, the sender protocol ceases to accept further messages from the application. Care is taken not to cause deadlocks with FIFO order.

Upon the arrival of a new message to a buffer, it is determined if any existing messages become obsolete and can later be purged by storing and updating a semantic index of the message buffer. By representing the obsolescence relation with a bitmap in the message, this index reduces to maintaining the latest messages stored in the buffer from each sender as a bitmap. Upon arrival, a logical “and” of both bitmaps (after the appropriate shift operations) is performed. Each remaining set bit represents a message that is known to be obsolete. Notice that iterating the resulting bitmap, even without resorting to bitmap operations existing on modern microprocessors, is reasonably fast as this bitmap is small. Messages known to be obsolete in a delivery buffer can be purged immediately. Purging the retransmission buffer requires that the safety of messages to be determined. This is achieved by running a global safety detection algorithm, which determines which messages have been queued for delivery by $f + 1$ processes and delaying purging accordingly. This algorithm is a modified version of the stability detection [14] algorithm requiring only $f + 1$ votes (instead of n) to make progress.

It may happen that a message is purged from all processes capable of retransmitting it before it has been delivered to all destinations. Processes that have not received a purged message must realize that this message can be removed from the negative acknowledgment queue and skipped. The mechanism used to do this is the stability detection algorithm. If a process purges a message from its retransmission queue, it fakes the stability of that message. Eventually, processes that have not received it discover that it has became stable, regardless of not having been locally received, and therefore conclude that the message is not missing but has been purged and should be skipped.

5.4 Centralized Simulation Model

The evaluation of the prototype implementation can be improved by using a centralized simulation model [1]. This combines a real implementation of the protocol code with discrete-event simulation models of the application and the network and has been shown to accurately reproduce timing properties of real systems. In detail, centralized simulation allows that single-threaded event-driven code to be run side-by-side with simulated system components. The execution of each event handler is timed using a profiling timer and the result used to update a simulated time-line.

By running all processes under control of the centralized simulation runtime within a single workstation, it becomes possible to obtain measurements with large numbers of processes to study the scalability of the protocol. It also becomes possible to perform observations that depend on a global clock, and, by stopping the clock, also to perform detailed accounting of various system parameters without disturbing the results. Centralized simulation has also simplified testing and debugging of the protocol implementation, by allowing the automation of regression tests with fault-injection. This configuration of the model has been validated both by micro-benchmarks for individual parameters (*e.g.*, overhead of the kernel network stack and scheduling latency) and by comparing results of protocol executions with results of the real system when possible (*e.g.*, distribution of round-trip times).

6 Performance Evaluation

In this section we study the performance of semantically reliable protocols and discuss how it is affected by different system, configuration and implementation parameters. Whenever possible, the values obtained from the analytical model are compared with the simulation results and with the data collected from the implementation. We concentrate on evaluating the performance of the protocol during steady operations (*i.e.*, when the membership is not changed).

6.1 Experimental Conditions

Simulations and prototype executions shown in the following section use traffic generated with constant intervals. Destination processes consume messages from the receiver queue also at a constant rate. To exercise system models and the prototype implementation we have selected the following pattern of message obsolescence: Message traffic consists of two distinct types of messages: *i) independent* messages that do not make other messages obsolete and that are not made obsolete by any other message; and *ii) overwrite* messages that obsolete their predecessors and are made obsolete by their successor with a given probability. The resulting probability distribution of D, the distance between related messages in the message stream, is characterized as follows:

$$f_{r,d}(x) = \begin{cases} 1 - r & \Leftarrow x = 0 \\ r(1 - \frac{r}{d})^{x-1}\frac{r}{d} & \Leftarrow x > 0 \end{cases} \tag{3}$$

This distribution is interesting because it is easily generated for simulation and because parameters r and d directly determine the characteristics of the traffic. The parameter r models the relative distribution of independent and overwrite messages: On the average, a ratio r of messages has overwrite semantics. Thus, r directly establishes an absolute upper bound on purging. The parameter d represents the diversity of overwrite messages, dictating the probability of two overwrite messages being related and thus sensitivity to buffer size N. With this distribution we can explore boundary conditions that limit the performance of our protocol.

Measurements of the prototype implementation were obtained with a network of 3 Pentium III/1Ghz workstations over a switched 100 Mbits Ethernet. One of the workstations is used as the sender. Another is used as a slow receiver, by sleeping an amount of time between deliveries. The sender and the third receiver do not introduce additional delays between deliveries, therefore consuming messages as soon as available. Unless otherwise noted the protocol used is FIFO S-RM. Initial measurements in each run are discarded, in order to obtain results only after the system is stationary. The operating system used in all workstations is RedHat Linux 7.1 and the protocol is compiled using the default GNU C++ 2.96 compiler with optimization turned on. As the ACE toolkit uses the `select()` system call for timing in Linux, the clock has a 10 ms resolution which limits the granularity of the sample. For instance, we used 10 ms as the period of the sender. Nevertheless, the low resolution of the timer prevents obtaining measurements for higher message rates despite the observer low processor and network usage.

Measurements using the centralized simulation model were obtained in one of the same workstations simulating groups with up to 64 elements. The performance of simulated processors is determined by the performance of the host processor. The model used for the network assumes 10 μs delay for each traversal of the UDP/IP stack within the operating system (as measured in Linux) and 100 Mbits duplex switched network. Applications do not consume processor time. We have not simulated the 10 ms granularity of Linux timers, which allows us to get finer-grained results. On the other hand, we modeled the scheduling latency of the operating system in order to be able to realistically reproduce round-trip measurements.

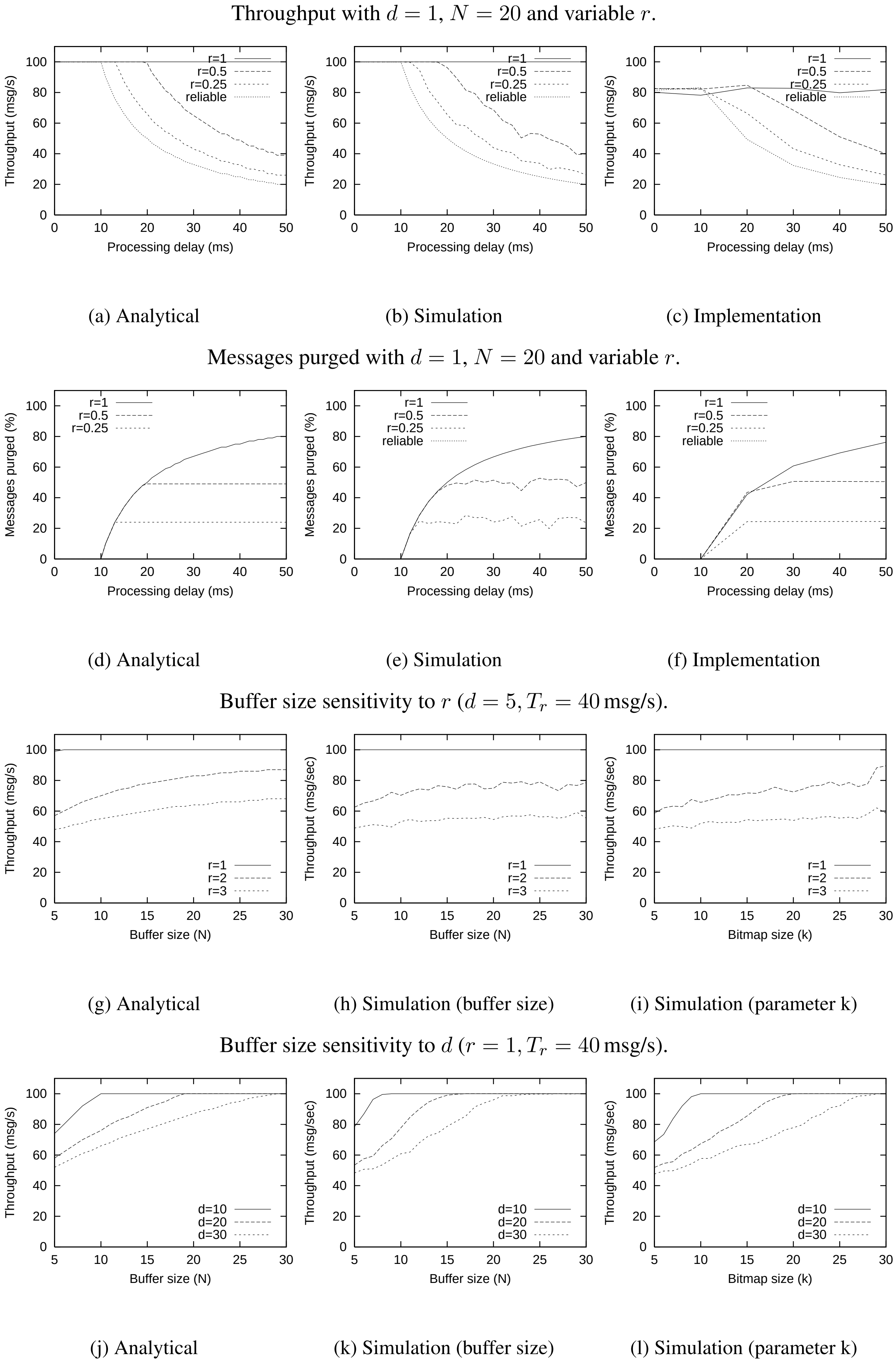
The prototype implementation is configured with bitmap size $k = 32$ and a buffer size of 40 messages at each node. A low amount of buffering is used to show that semantic reliability is practicable without large buffers as well as to reduce the time required to reach steady state required for measurements. As a consequence, stability and safety detection are aggressively configured with a period of 30 ms and a fanout of 3. In practice, larger buffer sizes and larger periods can often be used to reduce overhead.

6.2 Purging Efficiency

Throughput stability The effectiveness of purging protocol buffers, introduced by semantically reliable protocols, is measured mostly by the ability to accommodate a slower receiver within the group without disturbing the sender. This allows the resources of fast receivers to be fully used. Figure 5(a-c) presents the sustained incoming throughput (T_s) as a function of the processing delay of a single slow receiver. When purging is not applied, processing delays larger than 10 ms prevent a delivery throughput of more than 100 msg/s, thereby reducing the input that can be accepted. For instance, with 20 ms only 50 msg/s are accepted. By generating traffic with parameter $d = 1$ and a sufficiently large buffer size, parameter r directly determines the amount of traffic that can be purged. Using a semantically reliable protocol we observe that:

- When the amount of messages that can be purged is enough to accommodate the difference among T_s and T_r, the sender is undisturbed. For instance, with $r = 0.5$ half of the messages eventually become obsolete and can be purged. Therefore, the receiver can exhibit up to twice the delay (20 ms) without disturbing the sender.

- When the amount of messages that can be purged is not enough to accommodate the difference between T_s and T_r, such as with $r = 0.25$ and a delay of 20 ms, although the sender is disturbed the input allowed is still higher than without purging (*i.e.*, 75 msg/s



Throughput with $d = 1$, $N = 20$ and variable r.

(a) Analytical

(b) Simulation

(c) Implementation

Messages purged with $d = 1$, $N = 20$ and variable r.

(d) Analytical

(e) Simulation

(f) Implementation

Buffer size sensitivity to r ($d = 5, T_r = 40\,$msg/s).

(g) Analytical

(h) Simulation (buffer size)

(i) Simulation (parameter k)

Buffer size sensitivity to d ($r = 1, T_r = 40\,$msg/s).

(j) Analytical

(k) Simulation (buffer size)

(l) Simulation (parameter k)

Figure 5: Purging efficiency.

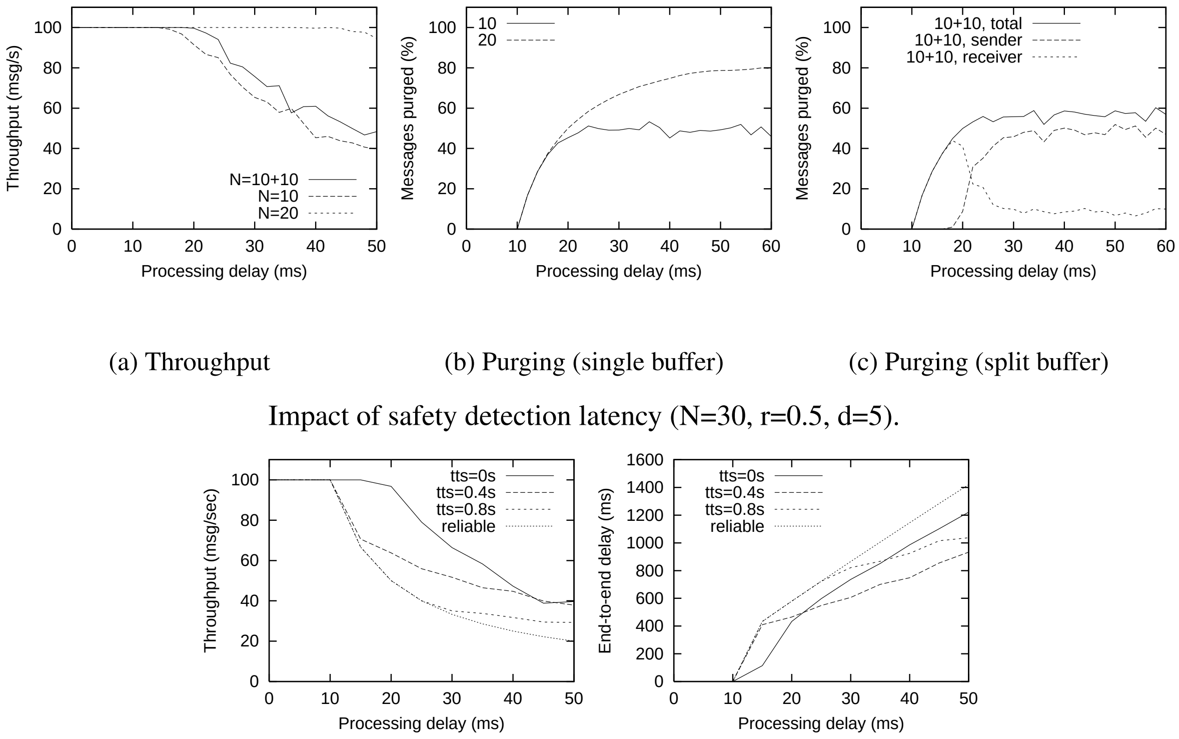
versus 50 msg/s).

These results are explained by the amount of messages that are purged and thus are not delivered to the slower receiver as shown in Figure 5(d-f). These results confirm that the maximum expected purging rate and consequent improvement in throughput can in fact be observed in practice.

Notice that with the prototype implementation it has not been possible to achieve an input of 100 msg/s. This happens because the test application tries to sleep for 10 ms between sending messages. However, due to the lack of accuracy of the operating system timer, it is often scheduled later, therefore reducing the input rate. This also means that measurements were obtained only for delays multiple of 10 ms which reduces the detail of Figures 5(c) and 5(f).

Configuration parameters The possibility of purging messages depends on recognizing pairs of related messages within a buffer. This is affected by the size of the buffer, the representation of the obsolescence relation and the characteristics of the traffic. Using simulation we can easily observe the behavior of the protocol when such parameters are varied. Figure 5(g-i) shows the impact of varying buffer size and maximum representable obsolescence distance k with several values of r and a low value for d. This makes the ratio of independent messages the limiting factor. Notice that the analytical model is somewhat optimistic. This happens because measurements were taken after the system is congested for a long time which means that buffers fill up with messages that never become obsolete. On the other hand, if the limiting factor is d, the diversity of traffic, all messages eventually become obsolete although related pairs of messages are far apart. As shown in Figure 5(j-l) the analytical model accurately describe the limitation that can be incurred when parameter k of the obsolescence encoding is too low. The analytical model is however too pessimistic when describing the impact of a small buffer. This is explained because in reality purging makes related pairs of messages which otherwise would be too far in the message stream closer, after purging other messages in between.

Buffer N=20 compared with N=10+10 (r=1, d=20).



(a) Throughput

(b) Purging (single buffer)

(c) Purging (split buffer)

Impact of safety detection latency (N=30, r=0.5, d=5).

(d) Throughput

(e) Delay

Figure 6: Implementation issues.

6.3 Implementation Issues

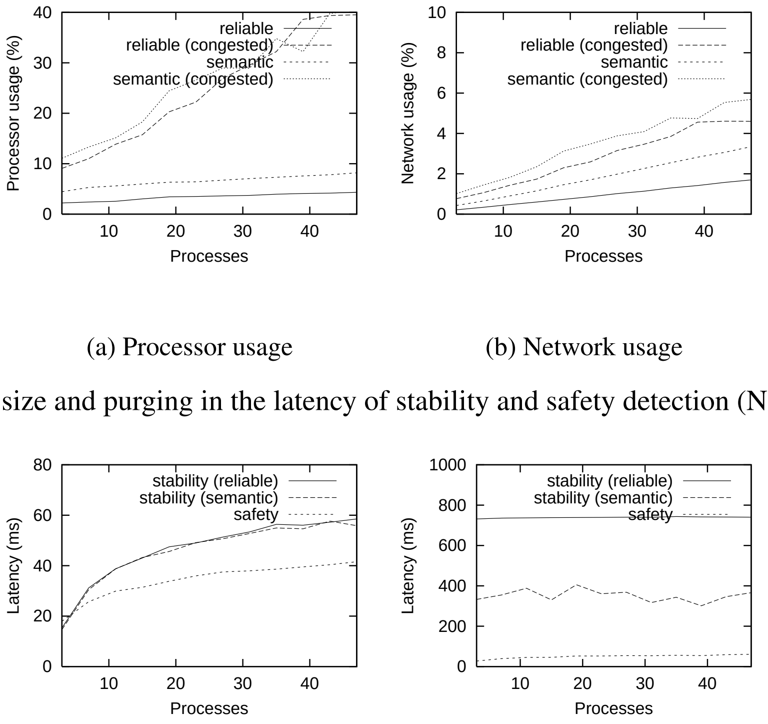
Eager and lazy purging In the analytical model we have also assumed that purging happens only when the buffer is full, thereby yielding a constant buffer size. This is a *lazy* purging strategy. In contrast, an *eager* purging strategy where purging is always applied can also be considered. Using the simulation model, it can be observed that similar purging rates and that neither results in messages being purged in the path to a fast receiver or when the system is not congested. However, buffer usage while purging is effective is lower with eager purging. Therefore eager purging results in better latency and better response to short congestion periods without otherwise impacting performance. As searching for obsolete messages does not represent a major overhead we consider only eager purging.

Single and split buffer Figure 6(c-e) shows simulation results for a scenario where both the sender and the recipients have a buffer size of $N = 10$ and purging is performed at both ends and compares this with both a single buffer of $N = 10$ and $N = 20$. Notice that, since congestion propagates back from the bottleneck, purging is first performed exclusively at the receiver until the buffer fills up with unrelated messages. After that, back-pressure is exercised and messages start being purged also at the sender side. Although the result is better than a single buffer, it is not comparable with a buffer with twice the size. Nevertheless, it is important to allow purging in both buffers as it allows the system to cope also with network congestion in addition to slower receivers.

Safety detection The requirement of determining safety prior to applying purging in retransmission buffers, which is implicit in FIFO Completeness, also affects purging, as messages can not be used for purging immediately as they enter the buffer. The resulting effect is similar to reduced buffer size, varying with the relation between safety latency and buffer latency. Figure 6(c) shows how throughput is reduced with an increasing safety detection time. Figure 6(d) presents end-to-end latency which is also greater as a consequence of increased buffer occupancy. Notice that when the system is congested, end-to-end latency becomes dependent on the throughput of the slower receiver, as any message entering the buffer has to wait for the consumption of all earlier messages before being itself delivered.

6.4 Resource Usage and Scalability

Processor and network usage Using the centralized simulation model it is possible to observe the behavior of the prototype implementation with a larger number of processes. Figure 7(a-b) shows processor and network usage of both protocols as measured at the sender with small messages (4 bytes of payload). Processor usage includes time spent both in the protocol and in the operating system, but excludes the application. When the system is not congested ($T_s < T_r$), most of the overhead of the reliable protocol is attributable to stability detection. When the system is congested ($T_s > T_r$ in general and $T_s = 3T_r$ here), most of the overhead at the sender is due to retransmitting messages and therefore grows with the number of processes involved. In both situations, the overhead of searching bitmaps for obsolete messages is negligible and is not presented. On the other hand, safety detection produces comparable overhead



Impact of group size and purging in resource usage (N=30, r=0.5, d=5, T_s=100 msg/s).

(a) Processor usage

(b) Network usage

Impact of group size and purging in the latency of stability and safety detection (N=30, r=0.5, d=5).

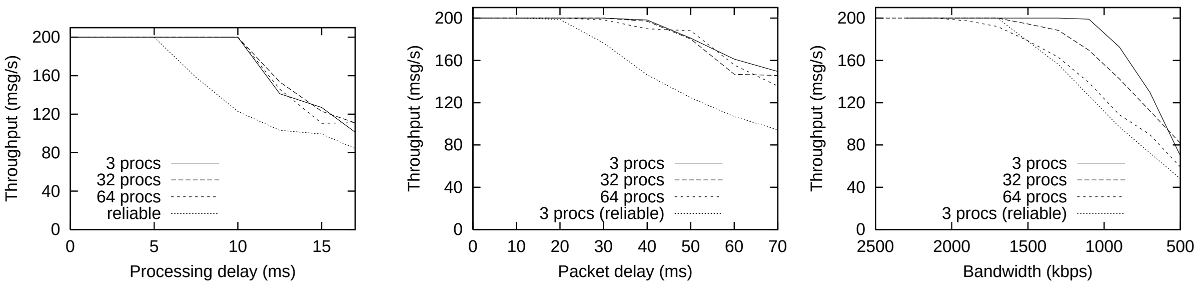
(c) No congestion ($T_s < T_r$)

(d) Congestion ($T_s = 3T_r$)

Figure 7: Resource usage and scalability.

to stability detection and thus represented a fixed amount of overhead regardless of the system being congested. It is thus interesting to consider disabling safety detection when the system is not congested, which is easily done by observing buffer occupancy. Network usage shows outgoing traffic only. As expected, when the system is not congested, safety detection doubles the overhead of the protocol. When the system is congested, traffic grows with the number of nodes due to message retransmission.

Safety detection Figure 7(c) presents the average latency of stability and safety detection when the system is not congested. As safety detection latency is quite large compared to stability detection latency, purging of retransmission buffers is harder. This is not critical, as when the system is not congested purging is only used to accelerate garbage-collection. Figure 7(d) presents similar results with a single congested receiver. Stability detection latency becomes dependent on the slower receiver. In this experiment safety detection is not affected because



(a) Processing delay upon delivery

(b) Packet delay in the network

(c) Network link bandwidth

Figure 8: Performance improvements with semantic reliability ($r = 0.5, d = 5$).

there is a single slow receiver, thus purging opportunities can be fully exploited and greatly reduce stability latency in the semantic protocol. This is true as long as the number of fast receivers is greater than f.

6.5 Discussion

We have analyzed the performance of our protocol using different approaches, namely using an analytical model, simulation and a concrete implementation. This experience allows us to draw conclusions about the validity of these approaches and identify the most relevant issues in the performance of semantically reliable protocols.

The overall effectiveness of the approach is summarized by Figure 8, which shows performance improvements with S-RM when compared to Figure 1. We have also observed that the results from the prototype are sufficiently close to the simulation model to allow reliable estimations to be extracted from simulations. In addition, results show that the simple analytical model is useful in predicting the behavior of the system from the characteristics of the traffic. Therefore it can be used by application developers as a configuration tool. This avoids having to use the more complex simulation model or configure the system by trial and error.

We have identified the following critical parameters in the performance of the protocol which have to be adjusted to the characteristics of the traffic: buffer size, maximum representable obsolescence distance and latency of safety detection. Configuration of the system can thus be done in two steps: *i)* a description of the traffic as a probability mass function of the distance between related messages is determined, either analytically or by profiling the application; *ii)* if probability of finding related pairs of messages accumulates in low values of distance, buffer size N and maximum obsolescence distance k can be selected as the minimum value which enables a sufficiently large share of related message pairs to be found. Safety detection latency is independent of the traffic and it was observed that it suffices to use the same configuration that was chosen for stability detection.

7 Case Study: Distributed Multi-Player Games

The purpose of this section is to illustrate the performance of the prototype under a real traffic pattern, in the current case, the traffic required to replicate the server of a distributed multi-player game. We believe that this is a relevant application scenario where stringent performance and consistency requirements meet. Although these applications are not typically supported by group communication services, this scenario is bound to change as the number of multi-player games hosted by commercial services as well as the number of players and spectators in each game are growing. Therefore it would be convenient to use standard protocols for dissemination of game information. Also as a result of this trend, long lived games have been appearing in an attempt to keep players loyal to a server. In such systems, the need to preserve the server state and offer continuous service becomes an important concern. Therefore, it is extremely relevant to find abstractions that ease the task of replicating this type of servers in an efficient manner.

The performance numbers presented here were obtained running the prototype implementation on the experimental setting described in Section 6.1. Before presenting the performance results we describe the traffic profile used for the experiment.

7.1 Traffic Characterization

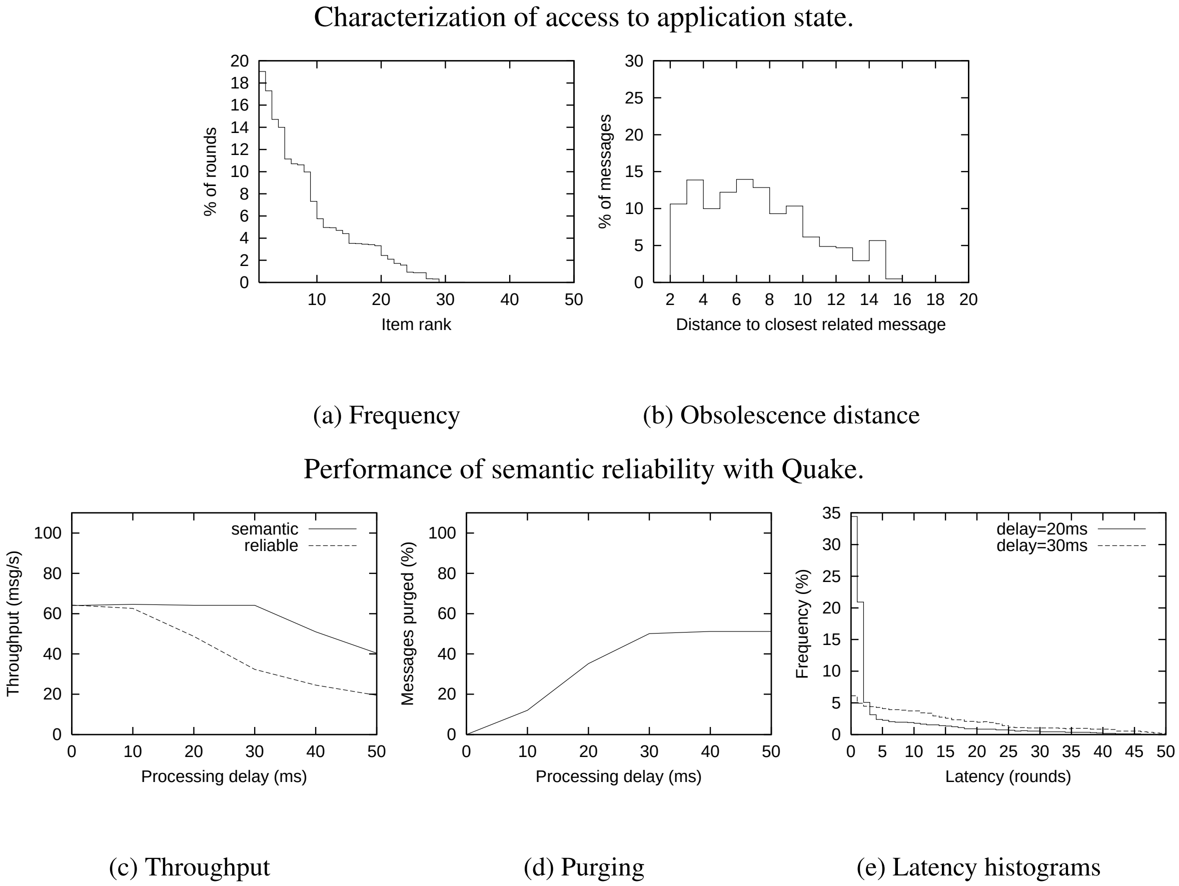
We have inspected the code of Quake$^{\text{TM}}$ [16], an open-source multi-player game, to extract concrete obsolescence relations. The state of the game is modeled as a set of items. An item is any object in the game with which players can interact. The background is described separately as it is immutable. Each item is represented by a data structure that stores its current position and velocity in the 3D space. The same data structure may also hold additional type specific attributes, such as the players remaining strength.

The game advances in rounds which correspond to frames that are displayed in players screens. Although the server tries to calculate 30 frames each second, this number can be reduced without loss of correctness. However, this degrades the perceived performance of the game hence the need to sustain a stable throughput. During each round the server gathers input from clients and re-calculates the state of the game. In each round, besides being updated, items can be created and destroyed. For instance, when a bullet is fired an item has to be created to represent it, and if a player is later hit, both the items of the bullet and the player have to be removed. The transmission of the updated state includes:

- Updated values of items, for instance, as their position is altered. These make previous values of updates obsolete as they convey newer values.
- Destruction and creation of items. These must be reliably delivered in order to ensure that items are kept consistent.

We have instrumented the game server to record obsolescence patterns from real gaming sessions. We detect which items are changed at each round by monitoring internal functions used to update the system state and to disseminate changes to clients. The results presented in this paper have been observed during a session with 5 players lasting for approximately 6 minutes and allowing us to record a total of 11696 rounds. This particular run was selected due to its length with a constant number of players.

From the traffic generated it was observed that a share of 41.88% of the messages never became obsolete. The obsolescence pattern of the remaining messages is related to the item update pattern. Although an average of 42.33 items were recorded active in each round, only an average of 1.39 items were modified. In addition, the results of Figure 9(a) show that a small number of items was modified frequently, while some items have not been modified at all during the measurement period. Therefore, consecutive updates of the same item are likely to be found close in the message stream. This is confirmed by Figure 9(b), which shows the



(a) Frequency

(b) Obsolescence distance

(c) Throughput

(d) Purging

(e) Latency histograms

Figure 9: Case study.

distribution of distance between related messages. Notice that related pairs are usually close together (often within 10 messages of each other). All items are represented by the same data structure, which is 52 bytes long. This is the size of message payload. Reliable messages have variable size depending on what happened in the round. Most were less that 32 bytes in size.

We have also collected data with other numbers of players. It can be observed that when more players join the game the message rate increases, the share of messages that never become obsolete decreases, but the distance between related messages increases. This suggests that higher purging rates would be possible that those presented here, although at the expense of larger buffer sizes.

7.2 System Configuration

This application is a good example of a case where the reliability constraints conflict with other system requirements, in this case, that of timeliness. The notion of message obsolescence may provide the means to achieve a reasonable trade off in this setting. Instead of introducing an arbitrary loss of messages, that could lead to loosing information about some entities, obsolescence allows to introduce a selective purge of messages during congestion periods.

Using the prototype implementation it is possible to observe the impact of purging in throughput stability. Figure 9(c) shows that semantic reliability allows the processing delay at the receiver can grow from 10 ms to 30 ms and be accommodated without impacting the sender. This is explained by being able to purge up to 60% of traffic as shown in Figure 9(d). No purging is observed by fast receivers, because messages are rapidly delivered before a substituting messages exists. Fast receivers are thus completely unaffected by congestion.

Although the application eventually receives a message that obsoletes every purged message, this information is received with some additional delay. This means that when semantic reliability is used to disseminate the information to clients, slower receivers will receive updates less often. We have measured this by counting the number of game rounds that an item is outdated until the substituting message is received. This metric avoids the requirement of synchronized clocks. When no purging exists, all modified items are updated within the round of modification. When the delay is 10 ms, 78% of modified items are updated within the same round and 94% within 10 rounds. Results when purging is higher are presented in Figure 9(e). Therefore, players with slower machines (or connections) will observe the game with less detail, which is clearly not as good has receiving all the information but certainly better than being excluded from the game or lagging behind the remaining players.

On the other hand, this effect is not a problem when the multicast protocol is being used to update backup servers for fault-tolerance. In fact, in this case all that matters is that eventually all servers are consistent, which is ensured by the protocol.

8 Related Work

To our knowledge, multicast protocols that address the issue of balancing high efficiency with adverse conditions such as congestion, variable message delays or network omissions rely on a mixture of accepting message loss and exploiting application-level semantic knowledge [5, 9, 2, 32].

The specific problem of ensuring stable throughput of reliable multicast has been addressed before [4, 5]. The proposed solution, Bimodal multicast, offers probabilistic reliability guarantees. In contrast, our approach is not probabilistic. Instead, we require the sender to selectively mark which messages can be purged by the protocol in overload conditions. Bimodal multicast does not require the sender to make this selection but requires the receiver to take corrective measure whenever a message is delivered to only some members of the group. If the loss compromises correctness the receiver may be forced to exclude itself from the group and rejoin later in order to get a correct copy of the state.

Application Level Framing [9] (ALF) requires the receiver to explicitly request retransmissions of lost messages that are considered relevant. As we have noted in Section 2, it may be hard to assess the relevance of a dropped message when its content is unknown. In the context of reliable process groups ALF seems to force too much complexity into applications, compromising the simplicity of the programming model. Our work is also inspired in the Δ-causal [2] and deadline constrained [32] causal protocols. These protocols however can only use real time to select which messages to purge, allowing timing constraints to be met at the cost of discarding delayed messages.

Message semantics has often been used to relax the ordering of messages. For instance lazy replication [19] relies on message semantics to relax causal order. Generic broadcast [23] is a relaxation of total order based on message semantics captured as a binary relation. It would be interesting to combine these approaches with our proposal. An analysis of the impact of relaxing ordering in the performance of multicast protocols is presented in [10]. Semantic information about messages has also been used for other purposes. For instance, the Bayou [36] replication system is sensitive to semantics of update messages. However, it relies on programs embedded in the updates which makes the implementation much more complex. In contrast, our proposal uses a simple mechanism that fits general purpose communication protocols. The work on Optimistic Virtual Synchrony [35] also uses semantic information to alleviate the cost of view changes by relaxing the ordering of views relatively to ordinary messages.

A primary-backup protocol which discards messages and provides real-time guarantees has also been proposed [37], although offering only a weak consistency model. In contrast, our proposal based on semantic reliability provides strong consistency [27] and a generic multicast primitive which can be used for other purposes other than primary-backup replication. The concept of semantic reliability in multicast protocols appears to be also useful outside the scope of group communication. Namely, it is being proposed for messaging in wireless networks [11].

9 Conclusions

Semantic Reliability is a novel correctness criterion for multicast protocols that makes use of the notion of message obsolescence. A message becomes obsolete when its content or purpose is superseded by a subsequent message. This paper offers an integrated perspective over a suite of protocols based on this notion.

The performance of the core aspects of these protocols is analyzed using three complementary approaches: an analytical model (whose purpose is to help application designers to quickly assess the impact of semantic reliability in their application), simulation (that allows to estimate the performance of the protocol in complex scenarios) and using a real implementation.

From this work we conclude that concrete applications requiring high throughput exhibit message obsolescence and that semantically reliable multicast protocols do in fact result in an improvement of throughput stability in spite of receivers with performance perturbations. Our work shows also how mechanisms of the implementation and configuration parameters affect the performance of semantic reliability, thus enabling effective deployment of these protocols.

References

[1] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *IEEE Intl. Symp. Reliable Distributed Systems*, 1997.

[2] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal. Efficient Δ-causal broadcasting. *Intl. J. Computer Systems Science and Engineering*, 13(5), September 1998.

[3] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Intl. Workshop Distributed Algorithms*, October 1996.

[4] K. Birman. A review of experiences with reliable multicast. *Software Practice and Experience*, 29(9), July 1999.

[5] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Computer Systems*, 17(2), 1999.

[6] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In S. Mullender, editor, *Distributed Systems*, chapter 8. Addison Wesley, 1993.

[7] A. Carzaniga, D. Rosenblum, and A. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Dept. Computer Science, Univ. of Colorado, January 2000.

[8] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. the ACM*, 43(2), March 1996.

[9] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM Symp. Communications Architectures and Protocols*, September 1990.

[10] C. Diot and F. Gagnon. Impact of out-of-sequence processing on the performance of data transmission. *Computer Networks*, 31(5), March 1999.

[11] S. Elf and P. Parnes. Applying semantic reliability concepts to multicast information messaging in wireless networks. In *IRMA Intl. Conf.*, May 2002.

[12] S. Floyd, Van Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6), December 1997.

[13] R. Guerraoui, R. Oliveira, and A. Schiper. Stubborn communication channels. Technical Report 98-278, Département d'Informatique, École Polytechnique Fédérale de Lausanne, 1998.

[14] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, Cornell Univ., Computer Science Dept., May 1998.

[15] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell Univ., Computer Science Dept., May 1994.

[16] Id Software Inc. Quake homepage. http://www.quake.com.

[17] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM Conf.*, August 1988.

[18] R. Jain. The art of computer systems performance analysis. *John Wiley & Sons, Inc.*, 1991.

[19] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. *ACM SIGOPS Operating Systems Review*, 25(1), January 1991.

[20] L. Lamport. Time, clocks and the ordering of events in distributed systems. *Comm. the ACM*, 21(7), 1978.

[21] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3), May 1994.

[22] R. Oliveira. *Solving consensus: From fair-lossy channels to crash-recovery of processes*. PhD thesis, Département d'Informatique, École Polytechnique Fédérale de Lausanne, February 2000.

[23] F. Pedone and A. Schiper. Generic broadcast. In *Intl. Symp. Distributed Computing (DISC)*, September 1999.

[24] J. Pereira. *Semantically Reliable Group Communication*. PhD thesis, Univ. do Minho, 2002.

[25] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast protocols. In *IEEE Intl. Symp. Reliable Distributed Systems*, October 2000.

[26] J. Pereira, L. Rodrigues, and R. Oliveira. Enforcing strong consistency with semantic reliability: Sustaining high throughput in reliable distributed systems. In P. Ezhilchelvan and A. Romanovsky, editors, *Concurrency in Dependable Computing*. Kluwer Academic Publishers, 2002.

[27] J. Pereira, L. Rodrigues, and R. Oliveira. Reducing the cost of group communication with semantic view synchrony. In *IEEE Intl. Conf. Dependable Systems and Networks*, 2002.

[28] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *IEEE Intl. Symp. Fault-Tolerant Computing*, June 1997.

[29] S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, May 1994.

[30] S. Raman and S. McCanne. Generalized data naming and scalable state announcements for reliable multicast. Technical Report CSD-97-951, Univ. of California, Berkeley, June 1997.

[31] M. Raynal and M. Mizuno. How to find his way in the jungle of consistency criteria for distributed shared memories (or How to escape from Minos' Labyrinth). In *IEEE Intl. Conf. Future Trends of Distributed Computing Systems*, September 1993.

[32] L. Rodrigues, R. Baldoni, E. Anceaume, and M. Raynal. Deadline-constrained causal order. In *IEEE Intl. Symp. Object-oriented Real-time distributed Computing*, March 2000.

[33] M. Satyanarayanan. A survey of distributed file systems. *Annual Reviews of Computer Science*, 4, 1990.

[34] D. Schmidt, D. Box, and T. Suda. ADAPTIVE — A Dynamically Assembled Protocol Transformation, Integration and eValuation Environment. *Concurrency: Practice and Experience*, 5(4), 1993.

[35] J. Sussman, I. Keidar, and K. Marzullo. Optimistic virtual synchrony. In *IEEE Intl. Symp. Reliable Distributed Systems*, October 2000.

[36] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *ACM Symp. Operating Systems Principles*, December 1995.

[37] H. Zou and F. Jahanian. Real-time primary-backup replication with temporal consistency guarantees. In *IEEE Intl. Conf. Distributed Computing Systems*, June 1998.