Semantically Reliable Multicast Protocols

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

Reliable multicast protocols can strongly simplify the design of distributed applications. However, it is hard to sustain a high multicast throughput when groups are large and heterogeneous. In an attempt to overcome this limitation, previous work has focused on weakening reliability properties. In this paper we introduce a novel reliability model that exploits semantic knowledge to decide in which specific conditions messages can be purged without compromising application correctness. This model is based on the concept of message obsolescence: A message becomes obsolete when its content or purpose is overwritten by a subsequent message. We show that message obsolescence can be expressed in a generic way and can be used to configure the system to achieve higher multicast throughput.

1. Introduction

The issue of achieving high and stable throughput in reliable multicast protocols has been addressed by several recent research efforts [4, 18, 2]. Two main impairments to support a sustained high throughput in this type of protocols have been identified: *i)* some protocols can be inherently non-scalable; *ii)* heterogeneous groups represent an hostile environment where any single slow-receiver can, due to the flow control mechanisms, become the bottleneck of the whole system.

The first problem has been addressed by the design of more scalable protocols that implement efficient mechanisms to disseminate messages and collect stability information [10]. The second problem is more difficult to tackle since no protocol can force a node to execute faster than its own resources allow. The problem can be circumvented by relaxing the reliability of multicast, for instance, by not delivering all messages to processes that are significantly slower than the majority of group members [4]. Unfortunately, when strong reliability is lost, most of the simplicity that was gained at the application level is also lost.

In this paper we propose a deterministic reliability model that makes use of message semantics to allow messages to be purged without compromising application correctness. The model is based on the concept of *message obsolescence*: A message becomes obsolete when its content or purpose is overwritten by a subsequent message. We show with practical examples that that obsolescence can be expressed in a generic way and used in different contexts.

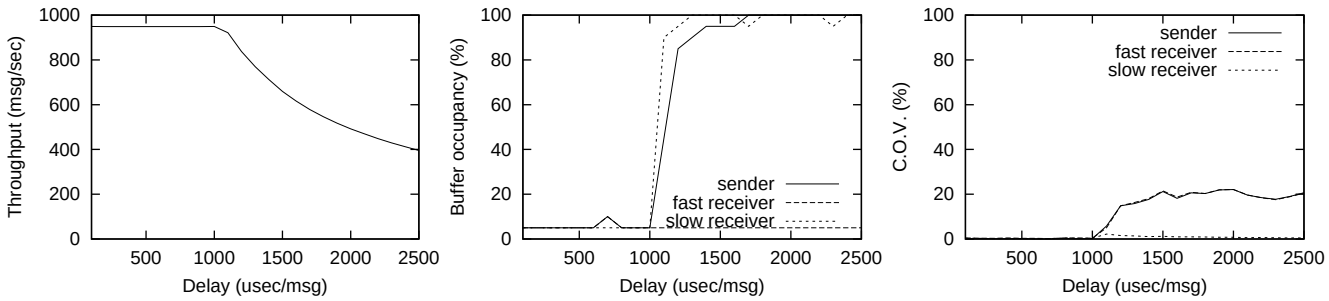
The paper shows that a reliable multicast protocol that purges obsolete messages can sustain higher throughput and discusses how the pattern of obsolescence that an application exhibits is related to different system parameters.

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 addresses our semantically reliable multicast protocol and, using both analytical and simulation models, shows how the protocol's performance can be assessed and related to traffic characteristics and system parameters. Section 5 illustrates the protocol using a concrete application. Section 6 compares our protocol with related work and Section 7 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 source, intermediate network links and routers, and sinks, 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 [12]. Workstation and server performance also suffer severe degradation when memory capacity is ex-

*This work was partially supported by the 234/J4 Franco/Portuguese Grant and by Praxis/ C/ EEI/ 12202/ 1998, TOPCOM.



(a) Throughput.

(b) Buffer occupancy.

(c) Inter-message variability (C.O.V. is the coefficient of variation, i.e., the ratio of the standard deviation to the mean).

Figure 1. Behavior of reliable multicast under load: One element of the group slows down by sleeping an increasing amount of time (x axis) between message deliveries.

ceed, a phenomenon known as thrashing [7].

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, for instance using the classical window-based mechanism as in TCP/IP [5]. Specifically, individual stages of the pipeline tolerate transient degradation periods of posterior stages 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.

To illustrate this behavior, we have simulated a simple reliable multicast protocol, using a small constant number of elements and an extension to multicast of the classical window-based mechanism [11]. This scenario allows us to concentrate on degradation due to flow control, without interference from phenomena such as ack implosion [8] that would surface in large groups. Further information about experimental conditions can be found in Section 4.4.

Specifically, we use one element of the group as the sender, producing messages at a constant rate. One other element is a fast receiver, consuming messages as soon as they are available. The third is the slow receiver, delayed by constant amount of time at the application level between two message deliveries.

Figure 1a shows the average throughput in messages per second (y axis) as measured leaving the sender for different delays introduced at the slow receiver (x axis). Notice that when the delay at the receiver is too big to keep up with the sender, flow control forces the sender to wait, thereby decreasing its throughput and affecting all receivers.

Figure 1b illustrates another inconvenient of the situation, showing average buffer ocupancy at the sender and at each receiver raising when the delay at the receiver forces the sender to slow down. In these circumstances transient performance degradation conditions within a single stage of the pipeline will immediately affect the whole system. For instance, the variability of the interval between messages grows because it becomes dependent on the retransmission mechanism. Consequently, variability of inter-arrival times at fast receivers is also affected, as depicted in Figure 1c.

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. Thus, the only definitive solution to this problem would be to replace the slowest component with a faster one. Unfortunately, transient problems by different machines may induce the same behavior as a consistently slow single node [3].

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, the application might be unable to take any corrective measure since it has no knowledge of that message's content.

This has lead to further research on providing some useful information to the application about messages that have been lost. For instance, 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 [16]. 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 the content and purpose of other messages sent in the past, therefore making them irrelevant. If obsolete messages have not been yet delivered to the application, they can be safely purged without compromising the application's correctness. In order to use this concept we must: *i)* identify which applications exhibit message obsolescence; and *ii)* show that it is possible to express this property in a generic form. In the remaining 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 [19]. Other examples include weakly consistent distributed shared memory systems, where memory operations are bounded by synchronization primitives to delay distribution of updates [17].

However, it is not always possible to implement these optimizations at the application level. If the distribution of updates is unpredictable and its dissemination has timing constraints, the application should forward the updates to the network as soon as possible. At that point, the message becomes out of reach of the application and cannot be discarded even if shortly after it becomes obsolete. Typical examples are applications such as on-line trading systems, where new quotes have to be continuously disseminated to a large number of recipients (a concrete example in given in Section 5).

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, Oliveira *et al.* [9] have formalized the notion of k-stubborn channel; a channel where reliability has to be ensured just for the last k messages (note that the number of rounds is not known *a priori*). The same authors have shown how the fundamental problem of distributed consensus [9, 13] can be solved in asynchronous distributed systems augmented with failure detectors and k-stubborn channels. A k-stubborn channel can be seen has a particular case of obsolescence.

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 formalize obsolescence as a relation on messages. For each pair of messages (m, m') in the relation, we say that the first, m, is *obsoleted* by the second, m'. The intuitive meaning of this relation is that if (m, m') is in the relation and if the system eventually delivers m', the application is correct regardless of m being delivered or not. We assume that this relation is transitive, anti-symmetric and coherent with causal order of events. One way to propagate obsolescence information is to tag each message with the identifiers of all messages that are made obsolete.

The expressiveness of this definition can be illustrated by a few examples. In a strictly reliable channel, no message can be discarded and the relation is empty. On the other extreme, a relation where every message obsoletes all preceding messages results in a $1-$stubborn channel [9]. A more complex example is presented in Section 5.

This definition is also generic as the message content needs not be known by the protocol implementation. It suffices to annotate each message upon multicast with the identifiers of previous messages that it obsoletes.

4. Semantically reliable protocol

Using information conveyed by the obsolescence relation, it is possible to modify a reliable multicast protocol in order to purge obsolete messages when the system is congested. In this section we establish what is the expected per-

formance of such protocol by presenting a simple analytical model for the efficiency of the purging procedure and test it by means of simulation. The analytical model can then be used to derive rules to properly configure the protocol in order to obtain the maximum possible throughput.

4.1. Purging obsolete messages

The protocol to purge obsolete messages is actually quite simple. The idea is that messages carry control information regarding the obsolescence relation. To prevent further overhead, when the system is not overloaded this information is not taken into account by the protocol and all messages are reliably delivered to the application.

Message obsolescence is only applied to prevent congestion. When buffer occupancy raises above a high-water mark the protocol scans the buffers for obsolete messages and purges them. Buffers are only parsed for obsolete messages when a large number of messages is stored locally, thus increasing the probability of finding related messages and effectively reducing buffer occupancy. As soon as buffer occupancy lowers, the protocol resumes reliable operation.

Over time, if enough messages can be purged the protocol will oscillate between reliable and congested mode without exercising back-pressure. If not, purging some messages at least ensures that back-pressure is weaker, minimizing upstream congestion. Note that the purging algorithm is activated first at the slower stage of the pipeline, the one whose buffers reach the high-water mark sooner. This may prevent other stages from becoming congested, strongly reducing the probability that alternating transient problematic receivers permanently congest all stages upstream.

Given the simplicity of the protocol, the interesting open issue is to understand which are the system parameters that affect the effectiveness of the approach and how these systems parameters can be related with the obsolescence pattern of the application's traffic. Our purpose is to obtain a model that allows the application designer to easily check whether our semantically reliable protocol allows higher throughputs to be sustained.

4.2. Analytical model

In order to assess the performance of our protocol, in terms of how different throughputs can be accommodated within the same group, we consider a simplified system model constituted by a single sender, a fast receiver and a slow receiver (see Figure 2).

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

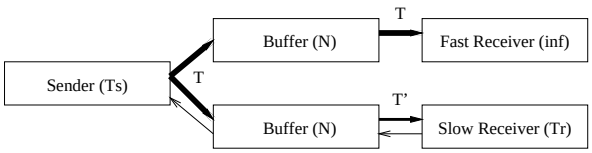


Figure 2. Simplified system model.

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.

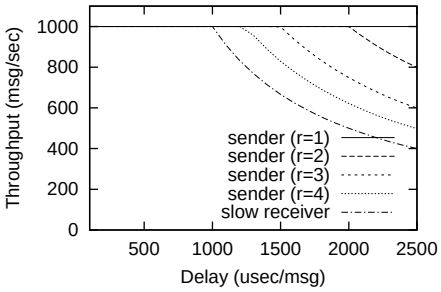
The probability of a message being obsoleted by a new message is thus given by $R_* = \sum_{x \geq 1} f(x)$, which is an estimate of maximum ratio of messages that can be purged by the protocol under continued congestion. However, this is not a good estimate of how the protocol would behave, as it implicitly 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. Total probability of an obsoleted predecessor existing in the buffer is thus $R = \sum_{x=1}^{N} f(x)$, 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 and T' (see Figure 2):

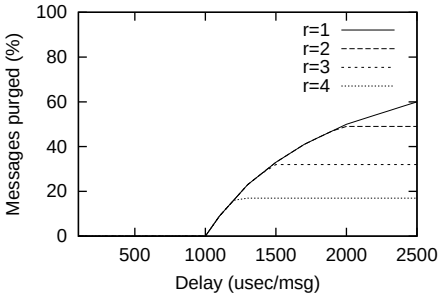
$$T = \min(T_s, \frac{T_r}{1 - R}) \quad (1)$$

$$T' = \min(T, T_r) \quad (2)$$

Naturally, if probability accumulates at low values of dis-

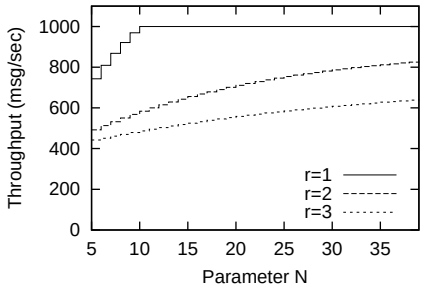


(a) Throughput.

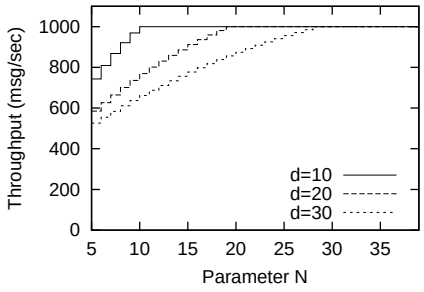


(b) Messages purged.

Figure 3. Despite the increasingly slow receiver, throughput at the sender remains unaffected for as long as obsolescence allows enough messages to be purged.



(a) Buffer size sensitivity to r ($d = 10$).



(b) Buffer size sensitivity to d ($r = 1$).

Figure 4. Depending on the obsolescence pattern of traffic, here determined by parameters r and d, buffer size N must be adjusted in order to obtain maximum throughput.

tance, *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.

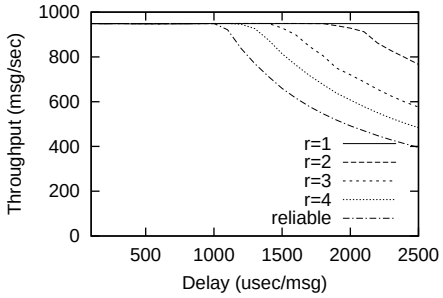
4.3. Applying the model

To exercise the model 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 distribution is characterized as follows:

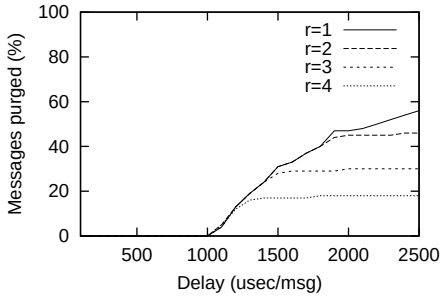
$$f_{r,d}(x) = \begin{cases} 1 - \frac{1}{r} & \Leftarrow x = 0 \\ \frac{1}{r}(1 - \frac{1}{rd})^{x-1}\frac{1}{rd} & \Leftarrow x > 0 \end{cases} \quad (3)$$

The parameter r models the relative distribution of independent and overwrite messages: On the average, one every r 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.

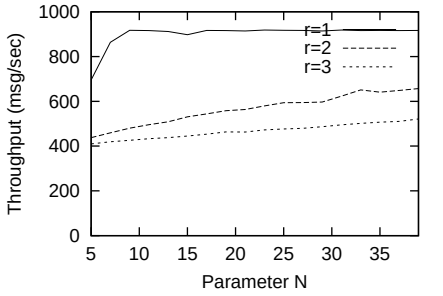
For instance, Figure 3 shows several aspects of protocol performance for this traffic pattern as predicted by our model. In particular, Figure 3a shows the expected behavior of a group where one element is increasingly slower but where traffic is distributed according to $f_{r,1}$ using large buffers. This result, should be compared to Figure 1a, where



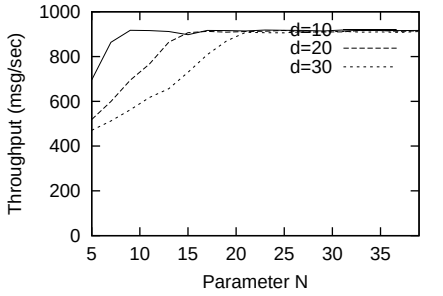
(a) Throughput with $d = 1$ and $N = 20$.



(b) Messages purged with $d = 1$ and $N = 20$.



(c) Buffer size sensitivity to r with $d = 10$ and $T_r = 400$.



(d) Buffer size sensitivity to d with $r = 1$ and $T_r = 400$.

Figure 5. Simulation results.

global throughput is limited by the slower receiver. Figure 3b explains why different throughputs are accommodated by depicting the amount of messages that are purged. Figure 4 shows the sensitivity to different buffer sizes of different combinations of parameters r and d in a situation where $T_s = 1000$ msg/sec and $T_r = 400$ msg/sec and thus messages must be purged in order not to impact overall performance.

This simple 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 only N messages are stored, they are likely not to be the last N messages. Furthermore, existing networks are not fully reliable and may deliver packets out of order. Thus, the actual distribution of messages in the recipient's buffers is even more unpredictable than considered above, where we assume that all messages are received in FIFO order. Thus the buffer might hold any N previous messages or even some posterior messages.

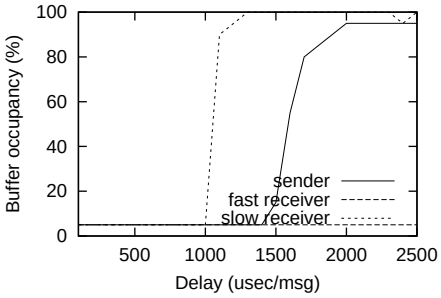
Additionally, in a real system we do not have a single buffer for each pair of sender-receiver nodes. Instead, we have two buffers, one at the sender and the other at the recipient, where purging may be applied. Naturally, if obsolete messages are purged in the sender's buffer, there are less chances that obsolete information reaches recipients. On the other hand, there is less load imposed downstream.

Although a detailed analytical model could be developed, simulations have shown that this simpler and easier to use model provides a good approximation of the system behavior, as described below.

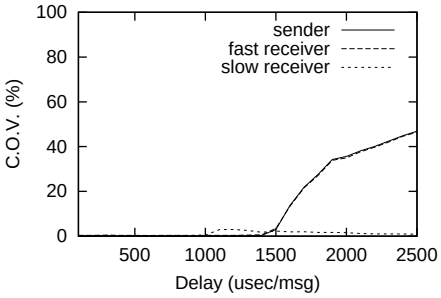
4.4. Simulation

In order to verify the validity of the analytical model and to study the impact of practical issues of protocol design, we resorted to simulation. In contrast to the analytical model, simulation allows us to consider the effect of message reordering and non-contiguous buffers.

We use a discrete event simulation model using real code for the protocol and a random model for the application and the network. This setup allows a precise simulation of the components of interest by using a highly accurate timer to measure the duration of the relevant events. The complexity

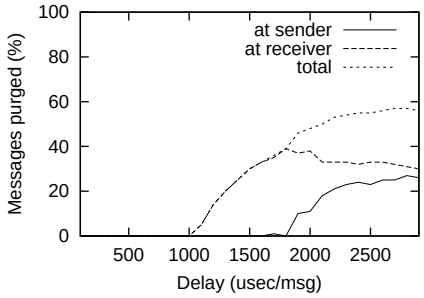


(a) Buffer occupancy.

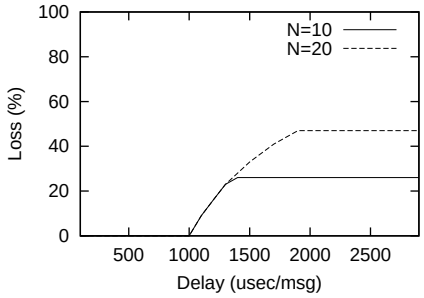


(b) Variability.

Figure 6. Simulation shows that both buffer occupancy and variability at the sender remain low despite congestion at the receiver.



(a) Simulation.



(b) Analytical.

Figure 7. Comparison of purging rates in a system configured with two buffers of size $N = 10$ with expected analytical results both for $N = 10$ and $N = 20$.

of the remaining system is abstracted as a random model for event duration. This approach has been shown to accurately represent real-time characteristics of the system being simulated while allowing centralized failure injection and omniscient observation [1].

A full-fledged reliable multicast protocol requires the combined used of several mechanisms. For instance, some form of negative or positive acknowledgement has to be used to mark messages as delivered. Inappropriate use of these mechanisms may lead to problems such as ack implosion [8]. In this paper we are interested in assessing the impact of message obsolescence and flow control, without being obfuscated by other aspects of protocol design. Thus, in these initial simulations we have chosen a small group of just three elements, such that buffer overflow is the only limiting factor in the protocol performance.

No process failures are assumed and thus no mechanism is used to change the membership of the group. Local network failure is also not considered. However, receive omission failures due to unavailability of buffer space are considered and used as an implicit back-pressure mechanism in the implementation of window-based flow-control.

Simulations shown in this section use traffic generated with constant intervals, mainly because jitter introduced by buffer overflow becomes easier to illustrate.[1] Destination processes consume messages from the receiver queue also at a constant rate. In addition, the obsolescence pattern of the traffic is generated according to the distribution introduced in the previous section and described by $f_{r,d}(x)$ (Equation 3), enabling direct comparison of results.

Figure 5 presents simulation results directly comparable to those of Figure 4. The observed purging rate with high values of N and d is higher than expected. This can be attributed to purged messages contributing to approximate related pairs of messages that otherwise would be too far to be found within the same buffer.

In addition, Figures 6a and 6b can be compared to their

[1] Later in Section 5 we show simulations using traffic generated with exponentially distributed intervals.

counterparts in Figure 1, showing that in the interval where purging is effective (*i.e.*, approximately between 1000μsec and 1500μsec) buffer occupancy and jitter at the sender remain low. Semantically reliable broadcast effectively decouples fast components from slow components in terms of congestion.

We now illustrate the difference between applying the purge procedure just at the recipient or both at the recipient and at the sender. Figure 7 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. 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. The result is approximately equivalent to a contiguous buffer when each half alone results in substantial purging. If not, the results are lower. Nonetheless, purging at both ends might still be useful for tolerating bottlenecks in different components of the system.

4.5. System configuration

These results support that when message obsolescence is taken into account higher throughputs can be sustained. The improvement is also directly related with the message obsolescence pattern and the amount of buffering accessible during purging.

Complete characterization of the application's obsolescence pattern can be achieved by profiling the application and deriving the probability mass function of D from observed frequencies.

Taking into consideration other factors such as available storage and impact in end-to-end delay, buffer size can then be determined in order to maximize the estimated purging rate under load given by R. Furthermore, this also permits evaluation of what is the maximum processing delay that can be tolerated before the source is affected by congestion.

5. Case-study

This section illustrates the configuration of the semantically reliable multicast to a concrete example. We show how our analytical model can be used to predict the system behavior and configure system parameters such as buffer sizes. Finally we show simulation data for the resulting system.

5.1. On-line trading system

As a case study we use an on-line trading system, more specifically, we study the publishing system that is used to disseminate information about operations and quotes to the traders's workstations. This system needs to sustain a high throughput to a large number of members [15].

Number of Stocks:	25	100	750	Total: 875
Frequency:	50%	40%	10%	

Table 1. Profiling information on updates: A small number of stocks is responsible for a large number of operations.

Both the timeliness and the reliability of the updates are extremely important in this context. Reliability is important because trader decisions are made based on available data and unreliable multicast may lead to the loss of critical information by some traders. Timeliness is also important because all traders, for fairness, should have the same information at approximately the same time. Unfortunately, when one of the recipients is congested, flow control can degrade the performance of the complete system. This is not acceptable and may force the exclusion of slow members [15].

Thus, this application is a good example of a case where the reliability constraints conflict with other system requirements (in this case timeliness) leading, in the worst case, to a complete denial of service during load peaks, ironically, when service is most valuable. The notion of message obsolescence may provide the means to achieve a reasonable tradeoff in this setting. Instead of introducing an arbitrary loss of messages, that could lead to some traders completely missing information about some stocks, obsolescence allows to introduce a selective purge of messages during congestion periods.

In the following, we assume that two consecutive messages containing information for the same stock are related, as the second obsoletes the first. This means that every trader always obtains the most up-to-date information about every stock, such as price and traded volume. Also, since purging of obsolete messages is performed first at the source of congestion, receivers with enough resources to sustain the throughput receive all operations despite the presence of some congested members in the group.

5.2. Obsolescence model of stock-trading

The distribution of trading operations by stocks has been reported to be highly skewed, such that a small subset of total stocks is accountable for a large subset of operations. Table 1 shows frequency data used in the design of a stock trading system [14]. Assuming that successive operations are independent, the probability mass function of the distance can be modeled using the geometrical distribution for

N	10	20	30
analytical	0.11	0.20	0.27
simulation with N	0.11	0.20	0.27

Table 2. Analytical and simulation results.

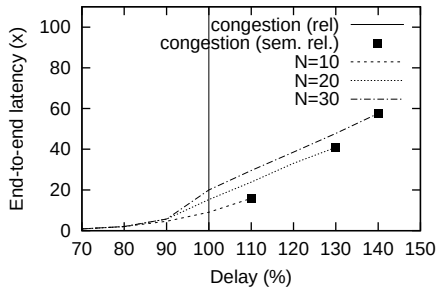


Figure 8. Results of simulation with different buffer sizes: Larger buffers sizes tolerate bigger delays without congestion at the receiver but result in greater end-to-end latency.

each frequency class. The resulting distribution is:

$$P(D=x) = \frac{0.25}{25}(1-\frac{0.5}{25})^{x-1} + \frac{0.16}{100}(1-\frac{0.4}{100})^{x-1} + \frac{0.01}{750}(1-\frac{0.1}{750})^{x-1}$$

It should be noted that in a real system, operations on the same stock tend to appear in bursts, even for infrequently traded stocks, due to fixation of the same trigger value for several operations and batching of small operations by market operators. If information about the same stock appears in bursts this augments the opportunity for purging obsolete information. However, we consider that each operation is independent and ignore this effect, resulting in a conservative prediction of efficiency.

5.3. System configuration

Given the obsolescence distribution described above, it is now necessary to find the ideal buffer size, considering both the optimization of throughput and end-to-end latency under load. Besides using traffic generated according to the previous distribution and with exponential inter-arrival times, experimental conditions are the same as in the previous section.

Although the application eventually receives a message that overwrites every purged message, this information is received with some additional delay. We name the interval between the sending of a message and the reception of that message or some other subsequent message that conveys the same information *semantic latency* and use it as a measure of quality of service. The final decision on what buffer size is chosen should take semantic latency into account. Normally, if the system is not congested, latency is limited only by transmission overhead and acceptance by the receiver. When the system is congested, buffers are full and thus any message might have to wait for all preceding messages in the buffer to be delivered, making buffer size relevant in semantic latency.

Results shown in Table 2 confirm the predictions of the analytical model: Purging is more effective for higher buffer sizes. In Figure 8 we present semantic latency, *i.e.*, interval between price fixation and subsequent notification of a slow receiver. The figure plots latency in terms of multiples of the average inter-arrival time (y axis) against receiver perturbation in percent of the same average inter-arrival time (x axis) and shows only stable system configurations, which are those that do not result in throughput degradation or exclusion of slow members. Notice that strict reliability would not allow stable configurations where any receiver is slower than the sender (*i.e.* to the right of the solid line) as the group would slow down or members would have to be excluded. For semantic reliability, the congestion points are depicted as a solid dots where purging is no longer enough to completely isolate the effect of slow receivers.

With a buffer size of $N = 10$ an increase of the processing delay in the order of 10% is tolerated; with a buffer size of $N = 20$, 30% increase is tolerated; finally, a buffer size of $N = 30$, allows an increase of the processing delay in the order of 40%. Notice that for the same receiver delay, the semantic latency increases with the buffer size, as increasingly older messages are selected for purging. However, this delay is a mild inconvenience when compared with the unpredictable delays that would result from throughput degradation at the sender.

6. 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 [4, 6, 2, 18].

The specific problem of ensuring stable throughput of reliable multicast has been addressed by Birman *et al.* [4]. 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 [6] (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 [18] causal protocols. These protocols use time to define obsolescence relations among messages allowing timing constraints to be met at the cost of discarding delayed messages.

7. Conclusions and future work

In the paper we have motivated and illustrated the advantages of using the notion of message obsolescence in the design of protocols for high throughput applications. The resulting protocol selectively purges messages that are consuming important system resources without compromising application correctness.

The paper has proposed a simple analytical model that enables reasoning about the efficiency of the protocol and the configuration of system parameters according to the obsolescence function of the target application. This model was validated through simulation. When applied to a traffic profile of an on-line trading system, our protocol is easily configured to allow a receiver to exhibit processing delays 40% higher than those required to process all messages in due time without disturbing the sender.

We draw the conclusion that semantic reliability is a viable approach to ensure global performance in the presence of perturbed group members. We are currently extending this work to study how the notion of message obsolescence interacts with other aspects of reliable communication, such as ordering constraints and membership.

References

[1] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *Symposium on Reliable Distributed Systems*, pages 35–42, 1997.

[2] R. Baldoni, R. Prakash, M. Raynal, and M. Singhal. Efficient Δ-causal broadcasting. *International Journal of Computer Systems Science and Engineering*, 13(5):263–269, Sept. 1998.

[3] K. Birman. A review of experiences with reliable multicast. *Software Practice and Experience*, 29(9):741–774, July 1999.

[4] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[5] D. Clark. RFC 813: Window and acknowledgement strategy in TCP. IETF Request for Comments, July 1982.

[6] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 200–208, Philadelphia, PA, Sept. 1990. ACM.

[7] P. Denning. The working set model for program behaviour. *Communications of the ACM*, 11(5), May 1968.

[8] A. Erramilli and R. Singh. A reliable and efficient multicast for broadband broadcast networks. In *Proceedings of the ACM workshop on Frontiers in Computer Communications Technology*, pages 343–352, Aug. 1998.

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

[10] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, Cornell University, Computer Science, May 1998.

[11] T. Hickey and R. van Renesse. Incorporating system resource information into flow control. Technical Report TR95-1489, Cornell University, Computer Science Department, Feb. 1995.

[12] V. Jacobson. Congestion avoidance and control. *ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988*, 18(4):314–329, 1988.

[13] R. Oliveira. *Solving consensus: From fair-lossy channels to crash-recovery of processes*. PhD thesis, École Polytechnique Fédérale de Lausanne, Feb. 2000.

[14] P. Peinl, A. Reuter, and H. Sammer. High contention in a stock trading database: A case study. *ACM SIGMOD Record*, 17(3):260–268, Sept. 1988.

[15] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange Trading System. In *Proceedings of The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing (FTCS'97)*, pages 309–313. IEEE, June 1997.

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

[17] 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 *Proc. of the IEEE Int. Conf. on Future Trends of Distributed Computing Systems*, pages 340–346, Lisboa (Portugal), Sept. 1993.

[18] L. Rodrigues, R. Baldoni, E. Anceaume, and M. Raynal. Deadline-constrained causal order. In *The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing*, Mar. 2000.

[19] M. Satyanarayanan. A survey of distributed file systems. *Annual Reviews of Computer Science*, (4):73–104, 1990.