Efficient Buffering in Reliable Multicast Protocols

Oznur Ozkasap, Robbert van Renesse, Kenneth P. Birman, Zhen Xiao

Dept. of Computer Science, Cornell University
4118 Upson Hall, Ithaca, NY 14853
{ozkasap,rvr,ken,xiao}@cs.cornell.edu

Abstract. Reliable multicast protocols provide all-or-none delivery to participants. Traditionally, such protocols suffer from large buffering requirements, as receivers have to buffer messages, and buffer sizes grow with the number of participants. In this paper, we describe an optimization that allows such protocols to reduce the amount of buffering drastically at the cost of a very small probability that all-or-none delivery is violated. We analyze this probability, and simulate an optimized version of an epidemic multicast protocol to validate the effectiveness of the optimization. We find that the buffering requirements are sub-constant, that is, the requirements shrink with group size, while the probability of all-or-none violation can be set to very small values.

1 Introduction

The aim of reliable multicast protocols is to provide all-or-none delivery of messages to all participants in a group. Informally, if any participant delivers the message, then eventually all participants should deliver the message. Since the sender may fail, processes buffer the messages that they receive in case a retransmission is necessary. Most existing reliable multicast protocols have all receivers buffer messages until it is known that the message has become stable (*i.e.*, has been delivered to every participant).

In such systems, it is always the case that the amount of buffering on each participant grows with group size for a combination of the following reasons:

1. the time to accomplish stability increases;
2. the time to detect stability increases;
3. depending on the application, the combined rate of sending may increase.

As a result, these multicast protocols do not scale well.

In this paper, we investigate optimizing buffering by only buffering messages on a small subset of participants, while spreading the messages over the entire membership. This way, each participant only requires a fraction of the buffering

---

[0] This work is supported in part by ARPA/ONR grant N00014-92-J-1866, ARPA/RADC grant F30602-96-1-0317, and the Turkish Research Foundation.

space used previously. Indeed, the amount of buffering space per participant may decrease with group size.

On the negative side, we introduce a small probability that a message is not delivered to all members. For example, this may happen if the entire subset responsible for buffering the message crashes before the message is delivered everywhere. We believe that in many situations such small probabilities can be condoned. In fact, epidemic multicast protocols such as used in the Clearinghouse domain name service [DGH+87], Refdbms [GLW94], Bayou [PST+97], and Cornell's bimodal multicast [BHO+99] already introduce such a small probability. Because of this, we focus our attention on using our suggested optimization in such protocols.

We investigate techniques for choosing suitable subsets of participants for buffering messages, ways for locating where messages are buffered in case a retransmission is required, how these techniques improve memory requirements, and how they impact the reliability of the multicast protocols. We take into account message loss and dynamic group membership. For analysis we use both stochastics and simulation.

This paper is organized as follows. In Section 2, we describe the group membership model, as well as how reliable multicast protocols are structured. The buffer optimization technique is presented in detail, and analyzed stochastically, in Section 3. Section 4 describes how this technique may be incorporated into an existing multicast protocol. In Section 5, we weaken our assumptions and describe a technique to improve the reliability of the optimized protocol without sacrificing the scalability. Simulation results are presented in Section 6. Section 7 describes related work, and Section 8 concludes.

2 Model

We consider a single group of processes or *members*. Each member is uniquely identified by its address. Each member has available to it an approximation of the entire membership in the form of a set of addresses. We do not require that the members agree on the membership, such that a scalable membership protocol such as [vRMH98] suffices to provide this membership information. We consider a non-Byzantine fail-stop model of processes. As is customary, recovery of a process is modeled as a new process joining the membership.

The members can send or multicast messages among each other. There are two kinds of message loss: send omissions and receive omissions. In case of a send omission, no process receives the message. In case of a receive omission, only a corresponding receiver loses the message. Initially, we assume that receive omissions are independent from receiver to receiver and message to message, and occur with a small probability P_{loss}, and that there are no send omissions. We weaken these assumptions in Section 5.

The members run a reliable multicast protocol that aims to provide all-or-none delivery of multicast messages, that is, to deliver each message to all processes that are up (not failed) at the time of sending. We do not require FIFO or total order on message delivery. All such protocols run in three phases:

1. an *initial (unreliable) multicast phase* attempts to reach as many members as possible;
2. a *repair phase* detects message loss and retransmits messages;
3. a *garbage collection phase* detects message stability and releases buffer space.

Most protocols use a combination of positive or negative acknowledgment messages for the last two phases. Epidemic multicast protocols accomplish the all-or-none guarantee with high probability by a technique called *gossiping*. Each member p periodically chooses another member q at random to send a gossip message to, which includes a report of what messages p has delivered and/or buffered. (Every message that is buffered by a process has been delivered by that process, but not necessarily vice versa.) q may update p with messages that q has buffered, but p has not delivered. q may also request from p those messages that p has buffered but q has not delivered.

Garbage collection in epidemic protocols is accomplished by having members only maintain messages in their buffer for a limited time. In particular, members garbage collect a message after a time at which they can be sure, with a specific high probability, that the gossiping has disseminated all messages that were lost during the initial multicast. This time grows $O(\log n)$, where n is the size of the membership as the corresponding member observes it [BHO$^+$99].

3 Basic Optimization

In this section, we describe the technique we use to buffer messages on only a subset of the membership. The subset has a desired constant size C. We say *desired* because, as we shall see, failures and other randomized effects cause messages to be buffered on more or fewer than C members. The subset is not fixed, but randomized from message to message in order to spread the load of buffering evenly over the membership.

We assume that each message is uniquely identified, for example by the tuple (source address, sequence number). Using a hash function $H : \mathit{bitstring} \rightarrow [0 \cdots 1]$, we hash tuples of the form $\langle$message identifier, member address$\rangle$ to numbers between 0 and 1. This hash function has a certain fairness property, in that for a set of different inputs, the outputs should be unrelated. Cryptographic hashes are ideal, but too CPU-intensive. CRCs (cyclic redundancy checks) are cheap, but the output is too predictable for our purpose: when given the 32-bit big-endian numbers 0, 1, 2, ... as input, the output of CRC-16 is 0, 256, 512, etc. We will describe a hash function that is cheap and fair, as well as why we require these properties, later.

A member with address A and a view of the membership of size n buffers a message with identifier M if and only if $H(\langle M, A\rangle) \times n < C$. We call a member that buffers M the *bufferer* of M. If H is fair, n is correct, and there is no message loss, the expected number of bufferers for M is C. Also, for a set of messages $M_1, M_2, \ldots$, the messages are buffered evenly over the membership.

If members agree on the membership, then any member can calculate for any message which members are the bufferers for this message. If members have slightly different memberships, it is possible that they disagree on the set of bufferers for a message, but not by much. In particular, the sets of bufferers calculated by different members will mostly overlap.

Also, if C is chosen large enough, the probability that all bufferers fail to receive a message is small. We will now calculate this probability. For simplicity, we assume that every member agrees on the membership (and is therefore correct), and that this membership is of size n. We consider an initial multicast successful if it is received by all members, or if it is received by at least one bufferer (which can then satisfy retransmission requests). Thus, the probability of success is the sum of the following two independent probabilities:

P_1: no members are bufferers, but they all received the initial multicast;
P_2: there is at least one member that is a bufferer and that received the initial multicast.

P_1 is simple to calculate, as, by fairness of H, "being a bufferer" is an independent event (with probability C/n), as is message loss (with probability P_{loss}):

$$P_1 = ((1 - \frac{C}{n}) \cdot (1 - P_{loss}))^n \tag{1}$$

P_2 can be calculated as follows:

$$\begin{aligned} P_2 &= P(\exists \text{bufferer that receives } M) \\ &= 1 - P(\forall \text{processes are not bufferers or lose } M) \\ &= 1 - P(\text{a process is not a bufferer or loses } M)^n \\ &= 1 - (1 - P(\text{a process is bufferer and receives } M)^n \\ &= 1 - (1 - \frac{C}{n} \cdot (1 - P_{loss}))^n \end{aligned} \tag{2}$$

The probability of failure P_{fail} is then calculated as:

$$P_{fail} = 1 - P_1 - P_2 = (1 - \frac{C}{n} \cdot (1 - P_{loss}))^n - ((1 - \frac{C}{n}) \cdot (1 - P_{loss}))^n \tag{3}$$

Assuming P_{loss} is constant (independent of n), it is easy to see that as n grows, P_{fail} tends to $\mathrm{e}^{-C \cdot (1 - P_{loss})}$. Thus, given the probability of receive emission, the probability of failure can be adjusted by setting C, independent of the size of the membership. P_{fail} gets exponentially smaller when increasing C.

In many cases P_{loss} is a function of group size, as it depends on the size and topology of the underlying network. For example, in a tree-shaped topology, messages have to travel over $O(\log n)$ links. If P_{ll} is the individual link loss, then $P_{loss} = 1 - (1 - P_{ll})^t$, where t is the average number of links that the message has to travel (t grows $O(\log n)$). Worse yet, receive omissions are no longer independent from each other. Thus, setting C in this case *does* depend on n. We discuss a solution to this problem in Section 5, and see how this affects the choice of C in Section 6.

4 Implementation

In this section, we discuss the design of the hash function H that we use, how we integrate our optimization with an epidemic multicast protocol, and how this affects the buffering requirements of the protocol.

As mentioned, the hash function H has to be fair and cheap. It has to be fair, so that the expected number of bufferers for a message is C, and so that the messages are buffered evenly over the membership. It has to be cheap, since it is calculated each time a message is received. Cryptographic hashes are typically fair, but they are not cheap. CRC checks are cheap, but not fair. We therefore had to design a new hash function.

Our hash function H uses a table of 256 randomly chosen integers, called the *shuffle* table. The input to H is a string of bytes, and the output is a number between 0 and 1. The algorithm is:

```
unsigned integer $hash = 0$;
for each byte $b$ do
        $hash = hash$ XOR $shuffle[b$ XOR least_signif_byte$(hash)])$;
return $hash/$MAX_INTEGER;
```

To integrate optimized buffering into an actual epidemic protocol such as bimodal multicast, we have to modify the protocol as follows. Previously, members satisfied the retransmission of a message out of their own buffers. With the optimization, if a member does not have the message buffered locally, it calculates the set of bufferers for the message and picks one at random. The member then sends a retransmission request directly to the bufferer, specifying the message identifier and the destination address. A bufferer, on receipt of such a request, determines if it has the message buffered. If so, it satisfies the request. If not, it ignores the request.

This improves the buffering requirements of the epidemic protocol as follows. In the original protocol, the memory requirement for buffering on each member grew $O(\rho \log n)$, where ρ is the total message rate and n is the number of participants (assuming fixed sized messages and fixed message loss rate) [BHO+99]. This is because the number of rounds of gossip required to spread information fully with a certain probability grows $O(\log n)$. In the modified protocol, the buffering requirement on each member *shrinks* by $O(\rho \log n/n)$, since C is constant.

5 Improvement

Up until now we have assumed that the only message loss was due to rare and independent receive omissions. In this section, we will suggest an improved strategy in order to deal with more catastrophic message loss, without sacrificing the advantageous scaling properties. The improvement consists of two parts.

The first part is to maintain two message buffers. The so-called *long-term* buffer is like before, in which messages are kept for which the corresponding process is a bufferer. The *short-term* buffer is a buffer of constant size in which all messages are kept in FIFO order as they are received. (Actually, the size of this buffer is linearly dependent on the message rate ρ, but independent of group size.) Both buffers can be used for retransmissions.

The second part involves an improvement to the initial multicast phase. The idea is to detect send omissions or large dependent receive omission problems, and retransmit the message by multicasting it again (rather than by point-to-point repairs). Such strategies are already built into multicast protocols such as bimodal multicast [BHO$^+$99] and SRM [FJL$^+$97]. Thus, the initial multicast phase is subdivided into three subphases:

1a. an unreliable multicast attempts to reach as many members as possible;
1b. detection of catastrophic omission;
1c. multicast retransmission.

For example, in a typical epidemic protocol this can be done as follows. Members detect holes in the incoming message stream by inspecting sequence numbers. They include information about holes in gossip messages. When a member receives a gossip with information about a hole that it has detected as well, it sends a multicast retransmission request to the sender. The probability of this happening is low in case of a few receive omissions, but high in the case of a catastrophic omission. The sender should still have the message in its short-term buffer to satisfy the retransmission request. Since these retransmission requests are only triggered by randomized gossip messages, it will not lead to implosion problems such as seen in ack or nak based protocols.

These two parts, when combined, lead to two significant improvements. First, they make catastrophic loss unlikely, so that the assumptions of the original basic optimization are approximately satisfied. Secondly, since most message loss is detected quickly, retransmissions will typically be satisfied out of the short-term buffer without the need for retransmission requests to bufferers. The long-term buffer is only necessary for accomplishing all-or-none semantics in rare failure scenarios.

6 Simulation Results

To validate our techniques, we have simulated bimodal multicast [BHO$^+$99] with and without our buffering optimization. For our simulations, we used the ns2 network simulator [BLE$^+$99], and multicast messages from a single sender. In all

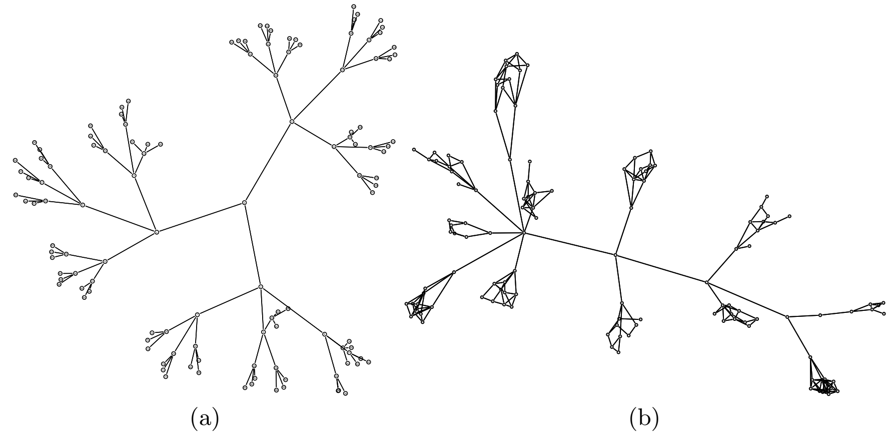

(a) (b)

Fig. 1. (a) a tree topology; (b) a transit-stub topology.

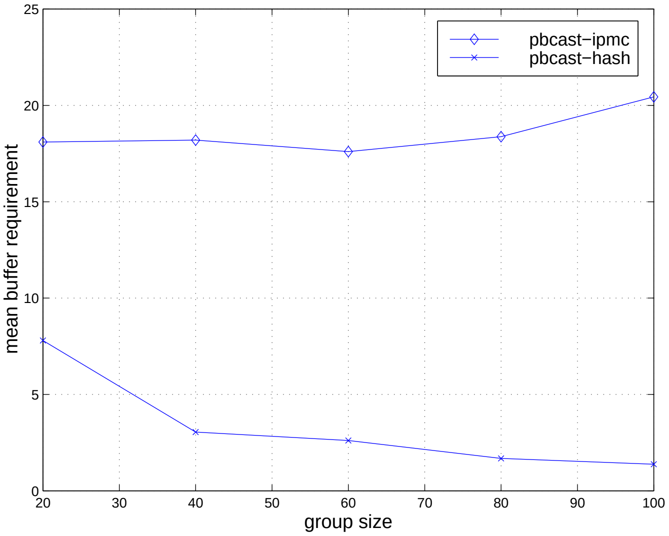
experiments, we set C so that $P_{fail} \approx 0.1\%$, based on the link loss probability and the number of members (see Equation 3).

We simulated on two different network topologies (see Figures 1(a) and (b)): a pure tree topology, with the sender located at the root of the tree, and a *transit-stub* topology (generated by the ns2 gt-itm topology generator), with the sender located on a central node. The transit-stub topology is more representative of the Internet than is the tree topology. The topology has an influence on the probability of message loss, but as we shall see, the overall effect of these two topologies on the buffering requirements is similar.

In Figures 2 and 3, we show the required amount of buffer space (the maximum number of messages that needed to be buffered) per member as a function of group size. In all these experiments, the individual link-loss probability in the network is 0.1%. In these cases, $C \approx 6$. The graphs for the original bimodal multicast are labeled "pbcast-ipmc," while the buffer optimized multicast graphs are labeled "pbcast-hash." Figure 2 uses a rate of 25 messages/sec. In (a) we used the tree topology, and in (b) the transit-stub topology. Figure 3 shows the same graphs for 100 messages/sec. We find not only that the buffering optimization greatly reduces the memory requirements on the hosts, but also that the buffering behavior is more predictable.

To see the effect of message rate and noise rates more clearly, see Figure 4. In both experiments we used a tree topology with 100 members. In (a), the link-loss probability is still fixed at 0.1%, while the message rate is varied from 25 to 100 messages/sec. In (b), the message rate is fixed at 100 messages/sec, but the link loss probability is varied from 0.1% to 1.5%. At $P_{ll} = 1.5\%$, we find that $C \approx 9$.

Figure 5 shows that the buffer optimization significantly reduces the memory requirements on each individual host, and also that the buffering responsibility is spread evenly over all members. Again, we are using a tree topology with 100



pbcast−ipmc
pbcast−hash
mean buffer requirement
group size
0
5
10
15
20
25
20
30
40
50
60
70
80
90
100