A Multicast Flow Control Protocol

Mark Hayden Zhen Xiao*

April 6, 1999

Abstract

This paper describes the goals, approach, implementation, and performance of MFLOW, a multicast flow control protocol. Our work focuses on the issues that arise from the increase in acknowledgments that occur as the size of the group grows. MFLOW makes use of bulk acknowledgments to substantially reduce the number of acknowledgments per multicast message. It also staggers acknowledgments from different destinations to avoid storms of messages. We demonstrate that staggering can both decrease the number of acknowledgments needed for effective flow control as well as increase the robustness of the protocol to changes in the environment. We analyze both message and computation efficiency of MFLOW. The protocol is computationally efficient and performs well in our experiments.

1 Introduction

A classical problem with point-to-point communication in distributed systems is that of flow control, or controlling the sender's transmission so that it will not congest the network or overflow recipients' buffers. This may occur when the sender is running on a lightly loaded computer while the recipient is running on a heavily loaded one so that the recipient is unable to handle all the messages that arrive, and, consequently, some of them may be dropped. This paper describes an approach to flow control in multicast (1-to-many) communication settings where there is a similar problem, except that in this case there is more than one recipient. Multicast communication is widely used in group communication systems, where groups of processes that act together for increased reliability, reduced response time, or better availability.

Two common approaches to flow control in point-to-point settings are rate-based and sliding window flow control [Tan96] [Guo98]. The approach with rate-based flow control is to put a restriction on the rate of the sender to slow it down sufficiently so that it will not swamp the receiver. However, getting a suitable rate takes some tuning: if the rate is too conservative, it wastes network bandwidth. The approach with window-based flow control is to have the receiver provide feedback to the sender. At a certain point, the sender must wait

---

*This work was supported by DARPA/ONR contracts N0014-91-1-10014 and ARPA/RADC F30602-96-1-0317.

for such feedback before sending more messages, thus giving the receiver a chance to keep up. In such protocols, the sender and the receiver each maintain a flow control window. The sender's window determines the amount of data it is permitted to send, and the receiver's window determines the set of messages it is permitted to receive. The sender only advances its window when an acknowledgment from the receiver arrives, thus effectively preventing the sender from sending more messages than the receiver can buffer. Acknowledgments are often merged, thus a single *ack* packet can acknowledge many messages.

The direction we take for multicast flow control is to generalize the sliding window protocol from pairs of processes to groups. In doing so, a couple of issues arise from acknowledgments as the size of the group increases. In point-to-point communication, only one acknowledgment is needed for the sender to forward its window. In multicast setting with n processes, $n-1$ acknowledgments are needed. The number of acknowledgment messages grows linearly with the size of the group, which means the system does not scale well. Another issue is that if all receivers respond at the same time, the burst of acknowledgments from multicast destinations may potentially drown the sender's buffer. Not only may some of the acknowledgments be dropped, but also the sender is periodically disturbed by bursts of acknowledgments.

We present a multicast flow control protocol, MFLOW, which addresses these problems. MFLOW uses bulk acknowledgments to substantially reduce the number of acknowledgments. The idea is that, just as in point-to-point communication where the acknowledgments are often merged, the acknowledgments in multicast applications can also be merged. Our protocol also explicitly staggers acknowledgments from multicast destinations. Our focus is on how to use the information available in group communication system (where the sender knows the buffer size at the receivers') to develop deterministic and efficient algorithms. It does not use any randomization or timeouts. In many applications such deterministic behavior is desirable. And it is computationally much simpler, which enhances its performance.

Multicast flow control is inherently more difficult than point-to-point flow control because more processes need to acknowledge messages. As the group size increases, if all destinations periodically reply with acknowledgments at the same time, the sender and the network can be flooded with occasional storms of acknowledgments. To prevent this, our protocol staggers over time the acknowledgments from different multicast destinations. This prevents the sender from being periodically disturbed by bursts of acknowledgments. Bursty behavior is bad in its own right, and can also be a cause of buffer overflow.

What is more, experiments show staggered acknowledgments make the protocol open to a wider settings of parameters, which makes it more flexible and suggests a greater robustness to changes in the environment than the non-staggered protocols. This appears to occur because the staggered protocol spreads acknowledgments out over time instead of sending them in bursts. In many situations the congestion in the network or the malfunction of some network routers or links may affect multiple receivers. Thus the omission of an acknowledgment from one receiver may imply other receivers have similar problems due to the current network condition. If all receivers respond at the same time, the sender can only detect such conditions periodically (when it finds that several destinations do not send acknowledgments) and may not be able to take appropriate action in a timely manner. In contrast, in a staggered protocol, the acknowledgments from different receivers are staggered over time and the lack of any single acknowledgment will eventually slow down the sender. Hence the sender can be constantly "informed" of the current state of the network and adjust its actions correspondingly.

Our protocol is efficient both in terms of computation and messages. We use an optimized implementation which achieves amortized $O(1)$ processing time on each message send and receive. Our protocol remains inactive when no application in the group is sending messages. Thus it does not waste network bandwidth when the group is computing or doing other tasks which do not involve communication. When applications do communicate, we show, for a given system configuration, how to precompute the optimal parameters in order to minimize message traffic overhead induced by the protocol. Finally, we present the performance of our protocol.

Our protocol is implemented as a protocol layer in the Ensemble system[Hay98]. Ensemble is an extensively layered and highly reconfigurable group communication system developed at Cornell University. It provides a library of protocols which can be dynamically linked into protocol stacks in various ways to meet the needs of different applications. Different layers may be substitutable for one another and may have different behavior under different workloads, which facilitates the comparison of different flow control techniques. The underlying principles of our design, however, can be applied to other architectures as well.

2 Goals and Approach

The goal of our research is to implement a multicast flow control protocol which effectively addresses the issues that arise from acknowledgments as the group size increases.

The MFLOW protocol uses credits[KM] to measure the available buffer space at the receivers. Each sender maintains a *window* which is used to bound the number of unacknowledged multicast messages a process can send. Initially, each sender keeps some amount of credit in stock. For each message it sends, the process deducts a certain amount of credit based on the size of the message. Multicast messages are transmitted only if the sender has enough credit for every destination. Otherwise, messages are buffered without being sent.

Every receiver keeps track of the amount of unacknowledged data it has received from each sender. When the amount exceeds a pre-specified threshold, the *ack_thresh*, it sends an acknowledgment to that process. On receipt of acknowledgment messages, the sender recalculates the amount of send credit, and buffered messages are sent based on the new credit. The protocol attempts to avoid situations where all receivers send acknowledgments at the same time, so that a sender is not flooded with acknowledgment messages.

Our approach has the following characteristics described in the sections below.

2.1 Independent streams

Every sender in the group is handled separately. For each of the senders, our protocol manages the flow from the sender to its multicast destinations. In a group of n processes, the protocol should be viewed as n different instances. The data structures that are used for managing the flow control for the different instances are all distinct. Effectively, the layers at all the processes are implementing the same "one sender, $n-1$ receivers" protocol for each of the processes in the group.

This may turn out to be a mixed blessing, however. On the one hand, it facilitates the analysis of the protocol because they are all independent. When analyzing the protocol, we only need consider the case of one sender. On the other hand, it ignores other information global to the group. Such information may be useful to optimize the performance. For example, some processes in the group may send more messages than others. Thus it may be reasonable to assign more credit to them. However, this would complicate our design without necessarily providing significantly better performance.

2.2 Modular implementation

In some systems, flow control is implemented along with other protocol features. For instance, the TCP protocol implements flow control, reliable transmission, and failure detection. This approach has several problems associated with it. The flow control aspects of the protocol become intertwined with the others. This in turn makes it difficult to change how flow control is managed without dealing with other issues that arise from the other parts of the protocol.

The Ensemble system, of which MFLOW is a part, uses layered protocol structure in order break large monolithic protocols into small micro-protocols. Protocols can be composed in a large number of ways to achieve differing sets of properties needed by different applications. Our experience has been that when high-level properties such as reliable multicast, failure detection, and flow control are decomposed into smaller layers, each of which implements several simple properties, they can be developed and tested more easily than a large, monolithic one. It also allows each protocol to be used in combination with a variety of other protocols, and facilitates experimentation with new communication properties and incremental extension of the system.

Our protocol requires reliable, FIFO multicast and point-to-point properties from underlying protocol layers. The "acknowledgment" messages it uses are not for reliability purposes (the reliable multicast protocol has its own acknowledgments in addition). They are only used to inform the sender about how much free buffer space the recipient has at the moment.

2.3 Weighted flow control

In making flow control decisions, this protocol attaches a weight to each message according to the size of the application data portion of the message (plus a fixed amount to represent the additional overhead of Ensemble protocol headers). The sender deducts the amount of credit according to this weight. The receivers keep track of the total size of multicast messages received from each sender. An acknowledgment is sent when the amount exceeds the acknowledgment threshold. Since larger messages consume more buffer space at the receiver's side than smaller ones, it makes sense to deduct more credit for them.

2.4 Event driven

Unlike many other flow control protocols, this protocol does not use timeouts for flow control. In other words, all actions of the MFLOW protocol are activated as the result of message send and receive events that are passed to this layer. This gives the protocol a relatively "deterministic" behavior. Also, the MFLOW protocol is inactive when there are no application messages being sent. Thus it does not waste network bandwidth when the group is computing or doing some other tasks which do not involve communication.

3 Message Efficiency

The additional acknowledgment messages from the multiple destinations in a multicast setting creates the concern that the acknowledgments may swamp the sender and the network, so it is useful to formally analyze the protocol to ensure its efficiency. The measure of efficiency we use is calculated in terms of the number of additional (non-application generated) messages the protocol introduces. Additional messages entail processing and communication overhead, which in turn slows down protocol stacks and impedes achieving high performance. Here we present such a formal analysis of the overhead and show how the protocol parameters can be set to achieve a balance between message efficiency and buffer size.

We define message efficiency to be the fraction of the total number of messages that were generated by the application. Although the analysis we present is calculated based on rates (msgs/sec) of message traffic, if you counted over a period of time *nmsgs* application messages and *acks* acknowledgment messages (the only additional messages that MFLOW introduces), the efficiency under our definition would be:

$$msg\_eff = \frac{nmsgs}{acks + nmsgs} \tag{1}$$

For instance, a protocol with no acknowledgments has efficiency 1; a naive protocol in which all receivers acknowledge every multicast message, the efficiency is $1/(n-1)$ (where n is the number of processes in the group), which is quite inefficient.

Our definition of efficiency is somewhat conservative because acknowledgment messages are usually smaller than application messages and therefore consume less network bandwidth and processing time. Thus, with an alternative definition of efficiency that considered bytes of acknowledgments verses application data would normally give higher levels of efficiency than presented here.

In analyzing the protocol, we make some assumptions about the behavior of the application and the network. We assume that the application is multicasting fixed size messages at a fixed rate. In addition we assume that the network has a bound on the typical message propagation delay. This analysis necessarily abstracts away some of the details of the system. However, not all of these assumptions need be exactly met by the system for the analysis to still hold most of the time. For instance, if some messages occasionally take longer than the expected delay time, then this would cause the protocol to not behave as predicted for a short period of time surrounding the aberrations.

The key to efficiency is to amortize the overhead of acknowledgments across many application messages. The MFLOW protocol uses the *ack_thresh* parameter to set the number of bytes a destination will receive before replying with an acknowledgment. A large *ack_thresh* increases the effectiveness of the amortization. The trouble with message efficiency is that it involves a tradeoff with the size of message buffers. In particular, the buffer size of the destinations (set by the *window* parameter) must be set at least as large as *ack_thresh*. The more frequent acknowledgments are sent, the smaller the buffers that are needed to maintain a particular rate of traffic. The opposite is also true, flow control protocols can be made more efficient through less frequent acknowledgments but with larger buffers.

In analyzing the protocol, we make use of the following parameters:

- *size*: the size of application messages.
- *rate*: the sender's rate of multicasting messages.
- n: the number of processes in the group.
- ϵ: the typical propagation delay of messages plus the interrupt and service time.
- *ack_thresh*: the number of bytes after which a destination sends an acknowledgment.
- *window*: the number of unacknowledged bytes a sender will transmit.

With these definitions, the number of *acks* in our protocol can be calculated as follows:

$$acks = \frac{nmsgs \cdot size \cdot (n-1)}{ack\_thresh} \tag{2}$$

Substituting Equation 2 into Equation 1 leads to:

$$\begin{aligned} msg\_eff &= \frac{ack\_thresh}{ack\_thresh + size \cdot (n-1)} && (3) \\ &= 1 - \frac{size \cdot (n-1)}{ack\_thresh + size \cdot (n-1)} && (4) \end{aligned}$$

To avoid introducing delays, the *ack_thresh* should be set so that the acknowledgments from multicast destinations can be expected to get back before the sender's window fills up, as shown in Figure 1 (we only depict one of the receivers).

After sending the message which exceeds the *ack_thresh*, the sender may continue for a time:

$$\frac{window - ack\_thresh}{size \cdot rate}$$

at which point it must stop if the acknowledgments have not come back yet[1]. Hence, in order for the acknowledgments to come back before they are needed, the *ack_thresh* should satisfy:

$$\frac{window - ack\_thresh}{size \cdot rate} \geq 2\epsilon \tag{5}$$

---

[1] The discussion here is simplified in that, after sending the message which exceeds the *ack_thresh*, the sender's remaining credits are a little less than $window - ack\_thresh$, depending on the size of that message.

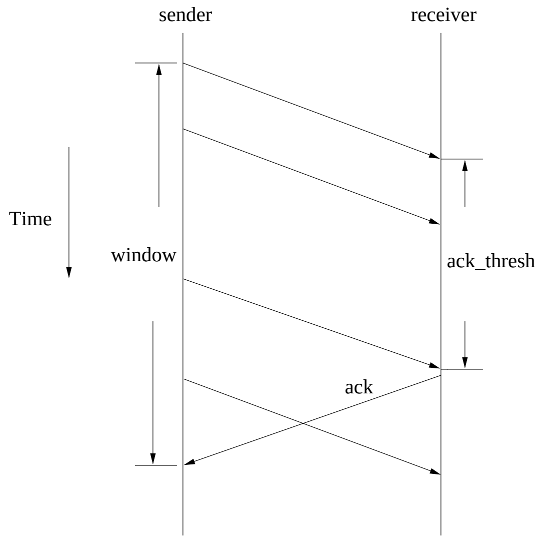


Figure 1: *Maximum value of ack_thresh without introducing unnecessary latency. The ack_thresh should be set so that the acknowledgments from multicast destinations are supposed to get back before the sender's window fills up. In the absence of message loss on the network or processing delay, this will prevent the MFLOW protocol from unnecessarily delaying messages.*

Solving this inequality yields:

$$\mathit{ack\_thresh} \leq \mathit{window} - 2\epsilon \cdot \mathit{size} \cdot \mathit{rate} \tag{6}$$

Combining Equation 4 and 6, we have:

$$\mathit{msg\_eff} \leq 1 - \frac{\mathit{size} \cdot (n-1)}{\mathit{window} - 2\epsilon \cdot \mathit{size} \cdot \mathit{rate} + \mathit{size} \cdot (n-1)} \tag{7}$$

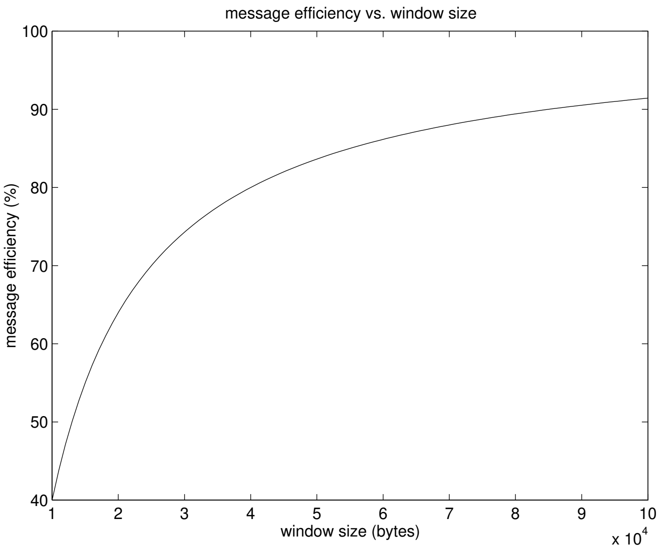
This is the maximum message efficiency we can get without introducing unnecessary latency. Figure 2 (a) depicts the growth of efficiency when the window size increases for a group of 10 processes with $\mathit{size} = 1K$ bytes, $\epsilon = 2$ milliseconds, and $\mathit{rate} = 1000$ msg/sec.

Also, from Equation 3 we can see that our protocol is more efficient for smaller messages than for larger ones.

4 Avoiding storms of acknowledgments

As mentioned previously, a concern in multicast flow control protocols is that storms of acknowledgments may overflow a sender's buffer if all receivers acknowledge at the same time. In addition to overflowing the buffer, the sender will be periodically perturbed by receiving all acknowledgments at once.

Figure 3 (a) depicts such a protocol in a group of 4 processes with $\mathit{ack\_thresh} = 15K$ bytes and $\mathit{window} = 30K$ bytes. One process in the group keeps multicasting messages at


message efficiency vs. window size
message efficiency (%)
100
90
80
70
60
50
40
1
2
3
4
5
6
7
8
9
10
window size (bytes)
x 10^4