A Reliable Multicast Protocol with Delay Guarantees

Nicholas F. Maxemchuk

Columbia University, Dept. of Elec. Eng., New York, N.Y.
nick@ee.columbia.edu

Abstract. The reliable multicast protocol guarantees that all receivers place the source messages in the same order. We have changed this protocol from an event driven protocol to a timed protocol in order to also guarantee that all of the receivers have the message by a dead line. In this work we present two modifications to the timed protocol that provide shorter deadlines. In the examples that we consider the tighter deadlines approach the nominal network delay.

1 Introduction

The Internet uses very simple protocols in the core of the network and relegates many functions to the end user. This strategy makes it possible to introduce new services by changing the programs at the users that require the services, rather than changing the entire network.

The Internet provides best effort delivery. It does not guarantee the message delay or that the message will be delivered at all. In order for the end user to guarantee that messages are delivered within a certain interval, the end user must have a concept of time and take action within the interval. In conventional ARQ protocols the source users a timer to periodically retransmit a message until it receives a response from the receiver. Alternatively, if the source transmits at known times, a receiver that has a clock and knows the source schedule can take action when messages aren't received. Periodic updates have been used in point to point transport protocols [1]. Recently, time has been added to the reliable broadcast protocol [2], RBP, to guarantee that all of the receivers have a message in a specified interval [3]. In the modified protocol messages are acknowledged according to a schedule and the receivers use absolute time to recover missing acknowledgements and source messages. Receivers that receive the acknowledgements and source messages do not have to send any further messages.

RBP was invented in 1984. This protocol used as few as one control message for each broadcast message, independent of the number of receivers, to guarantee that all of the receivers correctly received a broadcast message. In addition to guaranteeing that all of the receivers correctly receive every broadcast message, it guarantees that every receiver places the broadcast messages in the same sequence.

RBP was originally used to build a distributed database on an Ethernet[4]. In the early 90's, this protocol was adapted to operate on a multicast network over the Internet and was renamed the reliable multicast protocol[5], RMP.

E. Gregori et al. (Eds.): NETWORKING 2002, LNCS 2345, pp. 10–27, 2002.

RMP is event driven. The receivers do not take any action until a message is received. The protocol guarantees that all of the receivers "eventually" receive a message, rather than guaranteeing when they receive the message. If there are N receivers, the protocols guarantees that all of the receivers have a message after $N-1$ additional messages have been acknowledged. RMP is described in section 2.

In 1999 RMP was applied to an international, distributed stock market[3]. By adding a knowledge of absolute time to the protocol, and making the protocol time driven, rather than event driven, the earlier characteristics of RMP are maintained while also guaranteeing that every receiver receives every broadcast message within a specified time. The timed version of RMP, T-RMP, is described in section 3.

T-RMP periodically sends a control message that simultaneously acknowledges all of the unacknowledged source messages. All of the receivers know when a control message is scheduled to be transmitted and begin the recovery process soon after the scheduled transmission time, rather than waiting for a message. Once the control message is received, the receivers request any missing source messages that it acknowledged. When the period between control messages equals the average interarrival time of source messages one source message is acknowledged by each control message, on the average, and the message efficiency of T-RMP and RMP is the same. When the period between control messages is greater than the average interarrival time of source messages, more than one source message is acknowledged by each control message, and the efficiency of T-RMP is higher than RMP. However, as the period between control messages decreases, the message efficiency of T-RMP also decreases.

The version of T-RMP that is used in the stock market application is relatively easy to understand because the period between control messages is large enough for all of the receivers that have missed the control message or any of the source messages that it acknowledged to recover those messages before the next control message is transmitted. We can guarantee that the control message period is large enough for a receiver to recover a missing message because the ARQ protocol is not open ended. After a fixed number of attempts, the requesting site assumes that the site with the message has failed and enters a reformation process. Therefore, at the end of each control message period either all of the operable receivers have all of the acknowledged messages, or the system has entered a reformation process to identify failed sites.

The reformation process is a lengthy process. In order to prevent the protocol from performing a reformation when the network experiences slightly longer than normal delays, the message recovery time is much greater than the average message delay in the Internet.

The control message period is the guaranteed delivery delay for an acknowledged message. The delay guarantee that is provided by the original version of T-RMP is adequate for the stock market application, but reducing the delay will make the protocol applicable to a larger class of applications, such as remote classrooms where students ask questions.

One way to reduce the control message interval is to reduce the time between retry attempt to recover a lost message. As we make the retry intervals smaller we can take advantage of the small delays that usually occur in the network. However, the smaller retry intervals result in more frequent retries when a message has not been lost but is only delayed by the network. As the retry interval goes to zero, the time to recover a message can track the distribution of delays in the network, but the number of retries, and hence the number of overhead messages, becomes large. This effect occurs for all ARQ protocols that are used on the Internet, or any other network with variable delays. The effect is not unique to T-RMP and is not investigated in this paper.

In sections 4 and 5 we consider two ways to reduce the guaranteed delivery time that are unique to T-RMP. The original version of T-RMP uses separate retry counters to recover the control message and the source messages that it acknowledged. In section 4 we combine the counts and show that we can significantly reduce the control message period without increasing the probability of erroneously entering the reformation process. In the original version of T-RMP the control message interval, the time until a message is recovered by all of the receivers, and the time to enter the reformation process, are all the same. In section 5 we consider using different time intervals for each of these events. The operation of the protocol is more complicated. We show that the protocol operates as a D/G/1 queue and show, by an approximate analysis of the queue, that using different intervals fro the three events can significantly reduce the delay guarantees.

2 The Reliable Multicast Protocol

RMP has three characteristics that distinguishes it from earlier protocols:

1. Every receiver places the messages from the sources in the same sequence.
2. Every receiver eventually knows that every other receiver has the data.
3. When there aren't any losses, there is only one control message per source message, independent of the number of receivers. (In reference 6 there is an analysis of the number of messages that are transmitted when there are losses.)

The RMP protocol has two parts. The first part operates on multicast messages during normal operation. It guarantees delivery and ordering of the messages from the sources. The second part is a reformation protocol that reorganizes the broadcast group and guarantees the consistency of message sequences at the receivers after failures and recoveries. The complete protocol is described in reference 2. In this presentation we are concerned with the first part of the protocol.

There are n sources and m receivers that participate in the protocol, as shown in figure 1. The sources and receivers may be the same or different. A single receiver, called the token site, acknowledges a source message and assigns the message a sequence number. All of the receivers place the messages in the order indicated by the sequence number.

We guarantee that every receiver has all of the messages by sequentially passing the token to each receiver. A receiver does not accept the token until it acquires all of the preceding acknowledgments and the messages that they acknowledged. Therefore, when the receiver with the token sends an explicit acknowledgment for a source message, it implicitly acknowledges that it has received all of the source messages that have been acknowledged prior to this message.

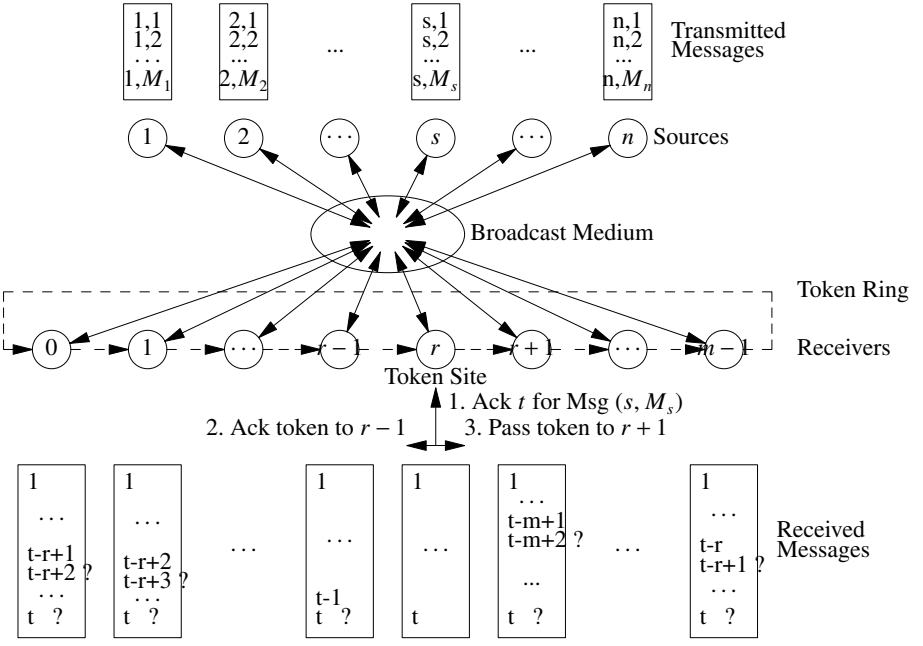

Fig. 1. The Reliable Broadcast Protocol

The sources use a positive acknowledgment protocol. A message from source s contains the label (s, M_s) to signify that it is the M_s^{th} message from source s. Source s transmits message M_s at regular intervals until it receives an acknowledgment or decides that the token site is not operating. If a source decides that the token site is not operating it initiates a reformation.

The receivers take turns acknowledging messages from sources by passing a token. A single control message, acknowledgment t from receiver r, serves three separate functions:

1. it acknowledges (s, M_s) and assigns it sequence number t,
2. it is an acknowledgment to receiver $(r-1) \bmod m$ that the token was successfully transferred to r, and,
3. it transfers the token to receiver $(r+1) \bmod m$.

The token transfer uses a positive acknowledgment protocol. Token site r periodically sends acknowledgment t until it receives acknowledgment $t+1$ or greater or it receives a separate token acknowledgment. If the acknowledgment isn't received in a specified number of attempts, receiver r decides that receiver $r+1$ is inoperable and initiates a reformation.

When r sends acknowledgment t it stops acknowledging source messages, even though receiver $(r+1) \bmod m$ may not have received, or may not be able to accept the token. This guarantees that at most one receiver can acknowledge source messages.

When a receiver accepts the token it also assumes responsibility for servicing retransmission requests. Receiver $(r+1) \bmod m$ does not accept the token until it has all of the acknowledgments and source messages that were acknowledged up to and including t. Receiver r does not stop servicing retransmission requests until it receives the acknowledgment for passing the token. This guarantees that there is always at least one site, that has all of the source and control messages, that is responding to retransmission requests.

Receivers place the messages in the sequence assigned by the acknowledgments. Each receiver, r, tracks t_r, the next acknowledgment that it expects. If an acknowledgment number greater than t_r is received, acknowledgment t_r is missing. If acknowledgment t_r is received and the source message that is acknowledged is not in the receiver's queue of unacknowledged messages, then the source message is missing. The receivers use a negative acknowledgment strategy. No control messages are sent unless a missing message is detected. When a receiver detects a missing message it recovers the message using a positive acknowledgment protocol. The receiver periodically requests the message until it receives the message or decides that the retransmit server is inoperable and initiates a reformation.

As the token is passed, the token site can infer information about the other receivers. When receiver r transmits acknowledgment t, receiver r and any receiver that receives the acknowledgment knows that

— receiver r has all of the acknowledged messages up to and including the t^{th} message,

— receiver $(r-1) \bmod m$ has all of the acknowledged messages up to and including the $(t-1)^{th}$ message,

— $\cdots$, and

— receiver $(r-m+1) \bmod m$ has all of the acknowledged messages up to and including the message acknowledged by $t-m+1$.

Since $(r-m) \bmod m = r$, receiver r knows that all of the receivers have all of the source messages up to and including the message acknowledged by $t-m+1$. By a similar argument all of the receivers know that all of the other receivers have all of the messages up to and including the $(t-m+2)^{th}$ message.

Figure 2 is an extended finite state machine, E-FSM, representation of the actions that a receiver takes when an acknowledgment is processed. The states indicate tests that are performed or situations where the receiver waits for an external stimuli, such as a message or a time out. The transitions between states are labeled with the event that caused the transition, followed by a "*"'ed list of actions that occur during the transition.

3 The Timed Reliable Multicast Protocol

T-RMP uses the same token passing mechanisms and retransmission strategies as RMP, as shown in figure 1. The difference is that T-RMP is time driven rather than event driven. Acknowledgments are transmitted by the token site at scheduled times separated by τ_t seconds. In addition, T-RMP is a bulk acknowledgment protocol. An acknowledgment message contains a list of all of the source message that the token site has received, but which have not been acknowledged by the previous token sites. The t^{th} token passing message acknowledges a sequence of k source messages, where k is variable. The messages are assigned sequence numbers $s+1$ to $s+k$, where s is the last sequence number assigned in the $(t-1)^{th}$ acknowledgment.

In T-RMP we assume that the receivers have synchronized clocks. Synchronization may be performed on the multicast network using other protocols [7, 8, 9] or may be performed on a parallel network, such as a satellite network, with deterministic delays. The clock synchronization technique is not part of T-RMP and is not considered in this presentation.

The primary advantage of the timed protocol is that a receiver detects a missing token based upon the time that it was scheduled to be transmitted, rather than later events that occur at undetermined times in the future. Negative acknowledgments have much more significance in the scheduled protocol than in the event driven protocol.

In the event driven protocol, RMP, we cannot assume that a receiver that has not sent

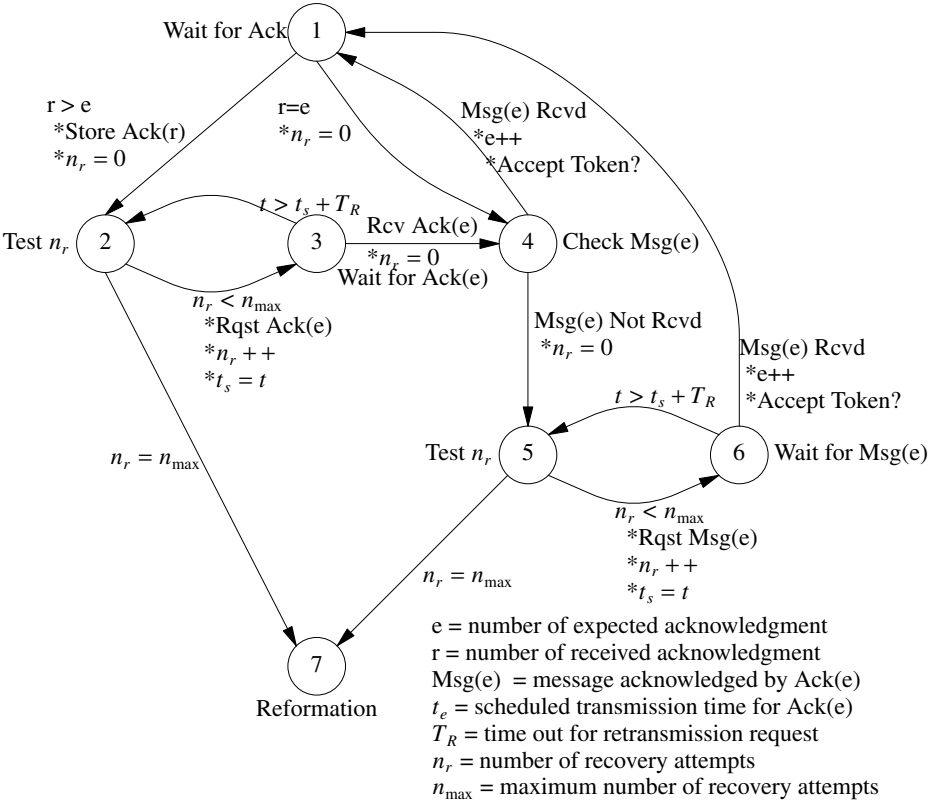

Fig. 2. An E-FSM representation of acknowledgment processing at a receiver in the RMP protocol

a negative acknowledgment has received a source message. The receiver may also have missed the positive acknowledgment for that source message and any subsequent acknowledgments that would indicate that it missed the first acknowledgment. We cannot be certain that the receiver has a source message until that receiver sends an implicit acknowledgment by sending a positive acknowledgment for a subsequent message.

In the scheduled protocol, T-RMP, a receiver is aware that it has missed an acknowledgment one network delay time after the acknowledgment is scheduled. Message recovery uses a positive acknowledgment protocol that retransmits unanswered requests at fixed intervals and declares a failure and places the system in reformation after a fixed number of unanswered requests. Therefore, after a fixed time following a message's acknowledgment, either all of the operable receivers have the message or the system is in reformation.

Figure 3 is the E-FSM representation of how acknowledgments are processed in T-RMP. We can use this state diagram to prove that all of the operable receivers have received a source message, or have placed the system in reformation within time $(n_{\max} + 1/2)T_R$ of when it was scheduled to be acknowledged. If the token site, that was scheduled to send the acknowledgment has failed, the system is placed in the reformation phase by the receivers. The sources don't have to detect a failed token site.

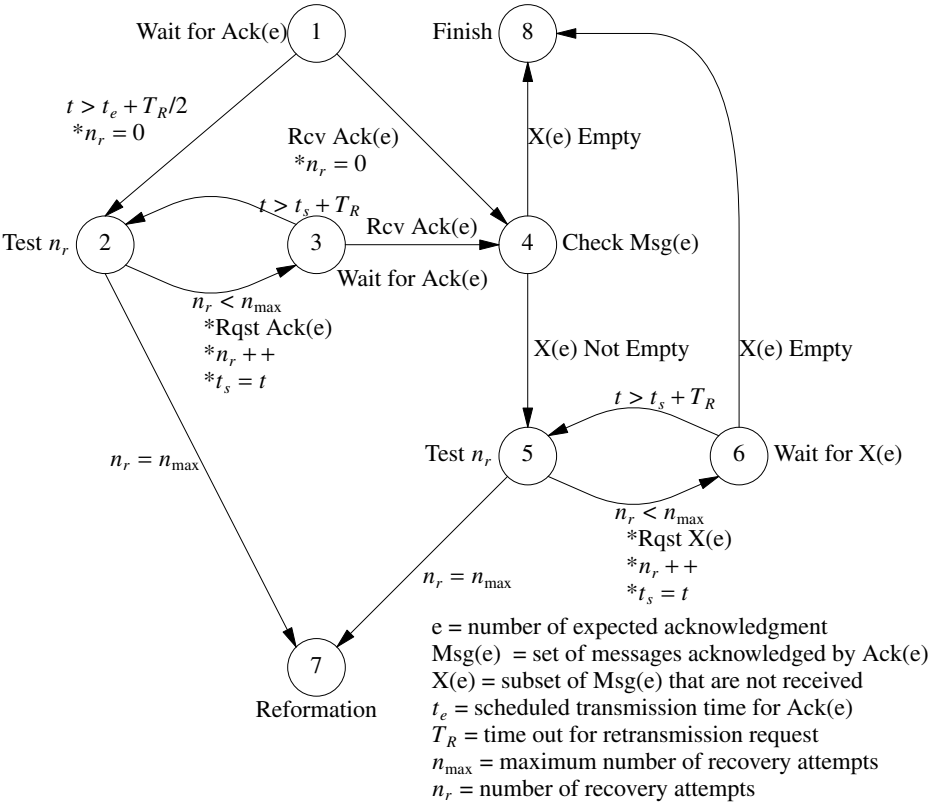

Fig. 3. An extended finite state machine representation of acknowledgment processing at a receiver in the timed RMP protocol

A source message is scheduled to be acknowledged at time t_e. If the acknowledgment is received before $t_e + T_R/2$, the receiver moves to state 4, with $n_r = 0$. Otherwise, at $t_e + T_R/2$ the receiver moves to state 2 with $n_r = 0$, requests the missing acknowledgment, increments n_r to 1, and moves to state 3. If the missing acknowledgment is received within T_R seconds, the receiver moves to state 4, otherwise it returns to state 2. The receiver circulates around the loop between states 2 and 3 at most $n_{\max}$ times. Either the receiver enters state 4 before

$t_e + (n_{\max} + 1/2)T_R$, or enters state 7, and initiates a reformation at time $t_e + (n_{\max} + 1/2)T_R$.

If a receiver enters state 4 at time t_4 such that $t_e + (k_4 - 1/2)T_R \le t_4 < t_e + (k_4 + 1/2)T_R$, then $n_r = k_4$. If the receiver has the acknowledged source message, then the receiver move to state 8, otherwise it moves to state 5. If $k = n_{\max}$, the receiver moves immediately to state 7, otherwise it follows the 5->6->5 recovery loop up to $n_{\max} - k$ times. The receiver enters state 7 at time $t_4 + (n_{\max} - k)T_R < t_e + (n_{\max} + 1/2)T_R$, if it does not enter state 8 prior to this time. Therefore, by $t_e + (n_{\max} + 1/2)T_R$ all operable receivers either have the message, or have started a reformation process. If the token passing period is $T_P \ge (n_{\max} + 1/2)T_R$, the next token site has recovered all of the messages, and is ready to acknowledge messages before the next acknowledgment is scheduled to be transmitted.

The structure of the state machine for T-RMP is similar to the state machine for RMP in figure 2. Two obvious differences are that:

1. T-RMP moves from state 1 to state 2 when the local clock exceeds the scheduled acknowledgment time plus a reasonable network delay, while RMP makes the same transition when it receives a token with a larger sequence number than expected, and,
2. T-RMP checks for, and may have to recover, a set of source messages for each acknowledgment, while RMP only checks for a single source message.

There are two other things that should be noted in the T-RMP state machine,

1. the time out that activates the transition from state 1 to state 2 is half the time out that activates the transitions between states 2 and 3 or 5 and 6, and,
2. the sum of retries to recover a missing acknowledgment and a missing message, is limited, rather than separately limiting the number of retries to recover each.

The sum of the timer delays in T-RMP determine how frequently we can transfer the token. The smaller the timers, the more frequently we can transfer the token. The more frequently we are able to transfer the token, the smaller the time until we are certain that all of the receivers have a message. In addition, smaller token transfer times result in a smaller waiting time until source messages are acknowledged. Therefore, we would like to make the total timer delays as small as possible.

4 Merged Retry Count

We merge the count of retry requests to recover lost acknowledgments and lost messages because it reduces the maximum time that we allow to recover messages, without increasing the probability of erroneously entering the reformation phase. As an example, consider a system with independent messages losses, P_L:

— The probability that a receiver does not receive an acknowledgment is $P_A = P_L$;

— The probability that the receiver misses at least one of k source messages that are covered by an acknowledgment is $P_S = 1 - (1 - P_L)^k$;

— And, the probability that the request for a retransmission from a receiver, or the retransmitted acknowledgment message, or retransmitted source messages, is lost is $P_R = 1 - (1 - P_L)^2$.

In a system that allows n_1 attempts to recover a missing acknowledgment and a separate n_1 attempts to recover any missing source messages, the probability of initiating a reformation process because a sequence of messages has been lost, rather than because a component has failed, is

$$P_{R,1}(n_1) = (P_A + P_S)P_R^{n_1} - P_A P_S P_R^{2n_1} .$$

In a system that allow a total of n_2 attempts to recover both the missing acknowledgments and retries, the probability of initiating the same erroneous reformation is

$$P_{R,2}(n_2) = (P_A + P_S)P_R^{n_2} + P_A P_S \left(n_2 P_R^{n_2-1}(1 - P_{R)} - P_R^{n_2} \right).$$

When $P_L \ll 1$, using a Taylor series expansion,

$$P_{R,1}(n_1) \approx \frac{k+1}{2} (2P_L)^{n_1+1} \text{ , and,}$$
$$P_{R,2}(n_2) \approx \left(\frac{k+1}{2} + \frac{kn_2}{4} \right) (2P_L)^{n_2+1} .$$

For $n_2 P < 1$, which is reasonable considering that $P_L \ll 1$,

$$P_{R,2}(n_1) > P_{R,1}(n_1) > P_{R,2}(n_1 + 1)$$

In other words, if we make the sum of the retries one greater than the number of separate retries to recover the acknowledgment and source messages, we are less likely to initiate an erroneous reformation process. A system that allows 3 separate tries to recover acknowledgments and source messages must allow 6 recovery intervals before passing the token. A system that monitors the sum of the retries can provide better performance while only allowing 4 recovery intervals before passing the token.

Of course we can make the above model more accurate by

— allowing different loss probabilities for different length messages, a short acknowledgment message versus up to k source messages,

— considering time correlation of the losses, and

— taking into account other receivers that may miss the same messages.

Our objective, however, is to demonstrate the advantage of summing the retry attempts, rather than to recommend a specific number of attempts for a particular network condition. In a real network the loss and delay change continuously. We

recommend increasing n_2 by one, and slowing down the token passing, when the receivers initiate unnecessary reformations, and decreasing n_2, and speeding up the token passing, when there is a long time between erroneous reformations. How long is long depends upon how badly we want to avoid erroneous reformations.

5 Separating Events

At each time t_e acknowledgment $Ack(e)$ is scheduled to be transmitted. $\Delta_T = t_{e+1} - t_e$ is the token passing period. Let $Ack(e)$ and the source messages that it acknowledges comprise the message set $Msg(e)$. At $t_e + \Delta_C$ all of the receivers that have recovered the source messages in $Msg(e)$ commit those messages. We assume that at $t_e + \Delta_C$ most, if not all, of the receivers have these messages. At $t_e + \Delta_R$ any receiver that has not recovered $Msg(e)$ initiates a reformation.

In our initial description of T-RMP $\Delta_T = \Delta_C = \Delta_R = \Delta_{init}$. This simplified the description and understanding of the protocol because the operation of the protocol is the same at every receiver and token site during every token passing interval. At each t_e, if the system is not being reformed all of the receivers, including the token site, have all of the $Msg(i)$ for *all* $i < e$. At t_e the token site transmits $Ack(e)$. At $t_e + T_R/2$ all of the receivers that do not receive $Ack(e)$ try to recover it. At $t_e + \Delta_R (\leq t_{e+1})$ any receiver that has not recovered $Msg(e)$ initiates a reformation process. Therefore, if the system is not being reformed, the operation at t_{e+1} is the same as the operation at t_e. In addition, t_{e+1} is the commit time for the messages acknowledged at t_e, since we can guarantee that all of the receivers have those messages.

When a source message is received at the token site it may wait up to Δ_T before the token is transmitted, and then must wait an additional Δ_C before the receivers commit the acknowledged message. We would like to make $\Delta_{\max} = \Delta_C + \Delta_T$ as small as reasonable, in order to provide stronger quality of service guarantees. In the initial system $\Delta_{max,init} = 2 * \Delta_{init}$. In this section we set $\Delta_T < \Delta_{init}$. However, in order to keep the probability that a receiver has a message the same as in the initial system, we must make $\Delta_C > \Delta_{init}$. We show that $\Delta_{\max} = \Delta_T + \Delta_C < \Delta_{max,init}$, for a certain range of Δ_T. We further reduce $\Delta_{\max}$ by making $\Delta_C < \Delta_R$. We justify this reduction by noting that false alarms, that cause unnecessary reformations, are generally more costly than the late arrival of a message.

When we make $\Delta_T < \Delta_R$ $Msg(e)$ may be recovered after $Ack(e+1)$ is scheduled to be transmitted, since $t_{e+1} < t_e + \Delta_R$. Recovering $Msg(e)$ after t_{e+1} does not have to affect the operation of a receiver that is not also the token site. The receiver can start recovering the missing components of $Msg(e+1)$ at the scheduled time whether or not is has completed the recovery of any $Msg(i)$, $i < e+1$. A receiver may have several recovery processes in progress simultaneously, or, since all of the requests for missing messages are directed to the current token site, the receiver may combine all of the requests into a single message.

However, when the site that is scheduled to transmit $Ack(e+1)$ fails to recover $Msg(e)$ before t_{e+1}, all of the receivers are affected. By the conventions of the protocol, the token site does not transmit an acknowledgment until it recovers the earlier messages and can service all retransmission requests. All of the other receivers may start transmitting their retransmit requests at $t_{e+1} + T_R/2$, but the recovery cannot start in earnest until after the token site completes its recovery and transmits the acknowledgment. In our model, the number of retries needed to recover messages, and the distribution of the recovery time, is independent of when the recovery starts. Therefore, if the recovery starts later than $t_{e+1} + T_R/2$, it will end later. If we make $\Delta_T < \Delta_{init}$, we must make $\Delta_R > \Delta_{init}$ order to keep the probability of reformation when there isn't a failure the same.

The operation of the token sites can be mapped onto the operation of a D/G/1 queue, where the period of the arrival process is Δ_T and the service process is the distribution of times to recover $Msg(e)$. In order to perform this mapping, site s_e, that transmits $Ack(e)$, arrives in the queue at time t_e. the scheduled time to transmit the acknowledgment. If site s_{e-1}, that transmits $Ack(e-1)$, has successfully transmitted $Msg(e-1)$ to s_e (that is to say, s_e has successfully recovered $Msg(e-1)$) before t_e, then the queue is empty, and immediately begins to service $Msg(e)$. The service time of $Msg(e)$ is the time needed to successfully transmit $Ack(e)$ from site s_e to site s_{e+1}, which is responsible for transmitting $Ack(e+1)$, and for s_{e+1} to recover any missing source messages in $Msg(e)$. If $Msg(e-1)$ is not transferred to s_e by t_e, s_e must wait for the transfer to be complete before beginning to service $Msg(e)$. Site s_e receiving the token at $t_e + \delta$ and beginning the next token transfer is equivalent to s_e arriving at the queue at t_e, and waiting until the previous service is completed at $t_e + \delta$ to begin its own service. Note that s_{e+1} begins trying to recover $Msg(e)$ at $t_e + T_{R/2}$, and combines any other missing messages with this request. This makes the service time independent of the past history of site s_{e+1}. Whenever s_e transmits the acknowledgment, s_{e+1} is ready to start recovery, without waiting for an earlier recovery to be complete.

The queue builds up because of the token passing process, but the waiting time distribution for the queue is the waiting time component for the delay at any receiver. None of the receivers can start recovering $Msg(e)$ until s_e has the token. Therefore they all have the same waiting time. The delay between the time that a source message is scheduled to be acknowledged and the time that a receiver has that message is the convolution of the waiting time distribution with the service time distribution. The service time distribution is the time needed for the receiver to acquire a message set $Msg(e)$, when the token sites have not failed. When the delay at a receiver reaches Δ_R, the receiver starts a reformation process, even though there has not been a failure. The waiting time is zero after a reformation. Since the probability of a false reformation is intentionally small, we approximate this probability as the probability of exceeding Δ_R in an infinite queue. The probability that a receiver has not acquired a source message when it is scheduled to be committed is the probability that the delay exceeds Δ_C.

Following the model in the previous section, the service time is

$$s = d_{n,1}(1 - x_A) + \frac{T_R}{2} x_A + j_1 T_R + d_{n,2} + d_{n,3} x_A + j_2 T_R + d_{n,4} + d_{n,5} x_S$$

where:
$d_{n,i}$ are delays through the network that depend on the source, the current token site and the network congestion,

$$x_A = \begin{cases} 1 & \text{with probability } P_L \\ 0 & \text{otherwise} \end{cases}$$

$$x_S = \begin{cases} 1 \text{ with probability } 1 - (1 - P_L)^r \\ 0 \text{ otherwise} \end{cases}$$

r is the number of arrivals from independent sources during Δ_T and is distributed as

$$p(r) = \frac{(\mu_A \Delta_T)^r e^{-\mu_A \Delta_T}}{r!}$$

and, j_i are the number of unsuccessful retransmission attempts before acquiring a missing message and is distributed as

$$p(j) = (1 - P_R) P_R^j \text{ for } j = 0, 1, 2, \ldots \text{ where } P_R = 1 - (1 - P_L)^2.$$

When the delay and retries are uncorrelated, the average service time is:

$$\mu_S = \mu_N \left\{ 1 + P_L + 2(1 - e^{-\mu_A \Delta_T P_L}) \right\} + \frac{T_R}{2} P_L + T_R \frac{P_R}{1 - P_R} \left\{ P_L + 1 - e^{-\mu_A \Delta_T P_L} \right\},$$

where $\mu_N = E(d_{N,j})$. When $P_L \ll 1$ and $\mu_A \Delta_T P_L \ll 1$,

$$\mu_S \approx \mu_N + P_L \left\{ \frac{T_R}{2} + \mu_N (1 + \mu_A \Delta_T) \right\}.$$

It's interesting to note that the time that it takes to transfer the token, s is a function of the token transfer period Δ_T and that μ_S decreases as the token transfer rate increases. When we transfer the token more frequently, fewer source messages arrive between token transfers, and it is more likely that we have not lost one or more messages. Therefore, when we transfer the token more often we are less likely to have to recover a source message. In the remainder of this section we are interested in the effect of decreasing Δ_T. In our first order analysis we will assume that μ_S is not a function of Δ_T. If we replace Δ_T with Δ_{init} ($\geq \Delta_T$), the value of μ_S will not decrease as we decrease Δ_T, and the actual advantage of decreasing Δ_T will be greater than we predict.

We do not know the service time distribution. The component of this distribution that

is the distribution of network delay between the receivers and token site is difficult to determine and changes. We will use a negative exponential distribution for the service time. This is reasonable because of the memoryless property of the process that recovers lost messages. More importantly, this assumption replaces the D/G/1 queue with a D/M/1 queue, which we know something about. At this point we will make a number of approximations in order to get a feel for the quantitative relationship between Δ_T, Δ_C and Δ_R.

The waiting time distribution in a G/M/1 queue[10] is:

$$W(y) = 1 - \sigma e^{-(1-\sigma)y/\mu_S} \quad \text{for} y \geq 0, \text{where,}$$
$$\sigma = A^*((1-\sigma)/\mu_S),$$

and $A^*(s)$ is the La Place transform of the arrival time distribution.

In our case the arrival time distribution is deterministic with period Δ_T, so that

$$A^*(s) = e^{-s\Delta_T} \quad \text{and} \quad \sigma = e^{-(1-\sigma)/\rho},$$

where $\rho = \mu_S/\Delta_T$. ρ is the utilization of the token passing channel. It if the fraction of the time the the token is in the process of being moved.

The equation for σ has one solution for $0 \leq \sigma < 1$, when $0 < \rho < 1$. This can be verified by considering the value of the exponential at $y = 0$ and $y = 1$, the slope at $y = 1$, and the second derivative over the range. The solution for sigma is plotted as a solid line in figure 4. Because of the shape of the curve, we approximate it with a quadratic. The least mean squared error fit is the quadratic:

$$\sigma = 1.168\rho^2 - .168\rho$$

The quadratic is plotted as the dashed line in figure 4. The fit is seen to be tight over the entire range.

The distribution of the delay is the convolution of the waiting time and service time distribution, and the probability of an erroneous reformation, P_{ref}, is the probability that the delay exceeds Δ_R. Therefore,

$$P_{ref} = e^{-(1+.168\rho-1.168\rho^2)\Delta_R/\mu_S}$$

In the initial system we expect the utilization of the token passing channel to be low because ρ is inversely proportional to Δ_T, $\Delta_T = \Delta_R$ and Δ_R is large enough that erroneous reformations occur infrequently. If ρ is small, the delay distribution is approximately equal to the service time distribution. This is a satisfying result for the initial system because whenever the waiting time is greater than zero, the system is put in reformation. The probability of reformation is approximately $P_{ref,init} \approx e^{-\Delta_{init}/\mu_S}$. The utilization is $\rho_{init} = \mu_S/\Delta_{init} = .43/\ln(P_{ref,init})$. If we adjust Δ_{init} so that $P_{ref,init} \leq 10^{-6}$, then $\rho_{init} \leq .07$, which justifies our claim that it is small.

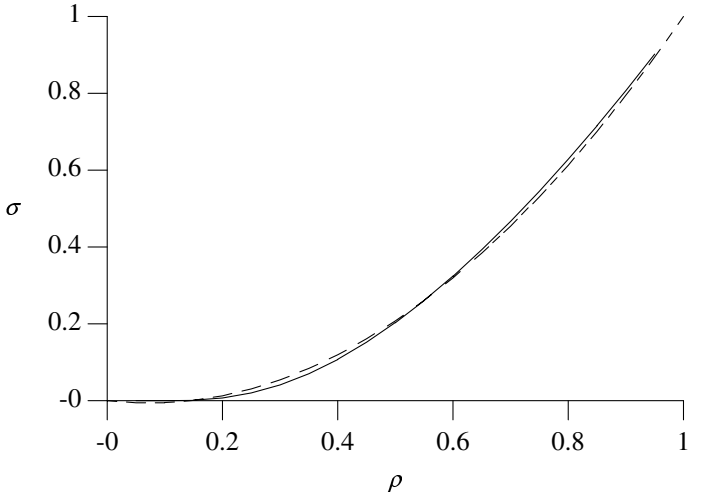

Fig. 4. Plot of $\sigma = e^{-(1-\sigma)/\rho}$ (solid curve) and $\sigma = 1.168\rho^2 - .168\rho$ (dashed curve)

Consider reducing $\Delta_T < \Delta_{init}$. ρ becomes larger. In order to keep P_{ref} the same, we must increase Δ_R so that

$$e^{-[1+.168(\mu_S/\Delta_T)+1.168(\mu_S/\Delta_T)^2]\Delta_R/\mu_S} = e^{-\Delta_R/\mu_S}.$$

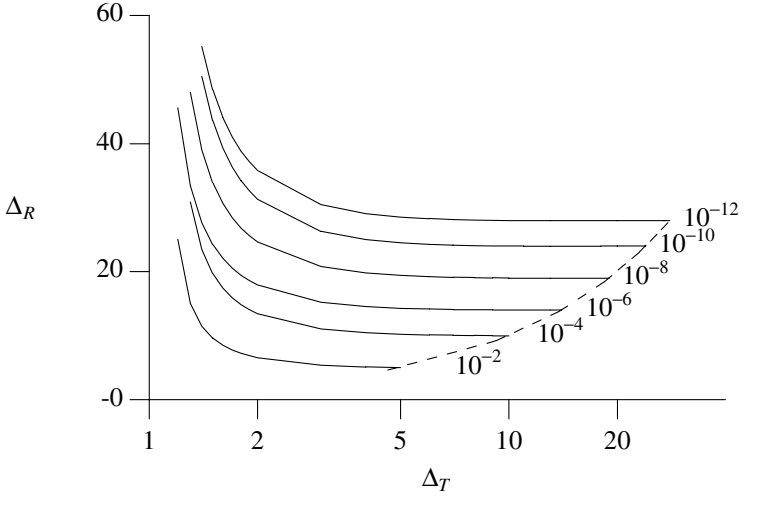

Fig. 5. Reformation delay Δ_R versus token passing period Δ_T for probabilities of erroneous reformation P_{ref} from 10^{-12} to 10^{-2}.

In figure 5 we plot equi-P_{ref} curves for P_{ref} from 10^{-2} to 10^{-12}. On the axes Δ_R and Δ_T are normalized with respect to μ_S, the average time that it takes to reliably transfer information between two participants in the protocol. The axis are dimensionless, and a value of 10 can be read as 10 average transfer times. The x-axis is also $1/\rho$. This axis is logarithmic with $\rho < 1$. The dashed curve is $\Delta_R = \Delta_T = \Delta_{init}$. We see that as we decrease Δ_T from Δ_{init}, Δ_R remains almost constant until ρ reaches about $30 - 60\%$, then grows rapidly as $\rho \to 1$. This graph shows us that there is almost no penalty for reducing Δ_T to $10 - 20\%$ of Δ_R.

Once a source message is received at the next token site, it may have to wait up to Δ_T until the next bulk acknowledgment message is scheduled to be transmitted, and then an additional Δ_C until the receivers use the message. The probability that a receiver has not acquired a message by Δ_C has the same form as the probability that the receiver has not acquired the message by Δ_R. The upper bound of this component of the message delay, $\Delta_T + \Delta_C$, normalized with respect to the message transfer time, is plotted in figure 6. The equi-probability lines are the probability that a receiver has not acquired the message by Δ_C. As Δ_T is reduced from Δ_{init}, the sum first decrease because Δ_C is increasing slowly. However, as $\rho \to 1$, Δ_C starts increasing quickly and the sum increases. There is a value of Δ_T that minimizes the sum, but the minimum is broad, so that the exact value of Δ_T is not critical.

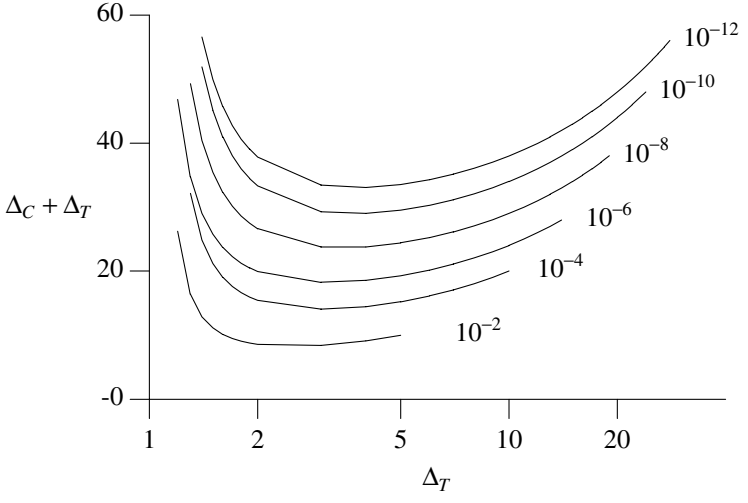

Fig. 6. Maximum delay from reception at token site to commit, $\Delta_C + \Delta_T$ versus token passing period Δ_T for probabilities that a receiver does not have a message by the commit time from 10^{-12} to 10^{-2}.

There are likely to be different penalties associated with a message arriving after the commit time and a system with the components operating properly entering a reformation. The quality of the information provided by a receiver may be adequate

if 10^{-3} or 10^{-2} of the messages arrive after they are scheduled to be used. However, in a system with 100 receivers, if each receiver places the system in reformation with probability 10^{-2} each time the token is passed, the system will be placed in reformation after most token passes, and will spend most of its time in reformation. Therefore, Δ_C and Δ_R should be selected separately.

Suppose that the system can tolerate 10^{-4} of the messages arriving after they are scheduled to be used. From figure 6, the minimum of $\Delta_T + \Delta_C$ is approximately 14, and is achieved when Δ_T is about 3. If we also require that $P_{ref} = 10^{-10}$, from figure 5, the delay until we enter reformation, Δ_R, is about 26, only 2 greater than it was for Δ_{init}, as noted by the dashed line in figure 5. If we try to meet both constraints with $\Delta_C = \Delta_R$, $\Delta_T \approx 4$, $\Delta_T + \Delta_C \approx 29$, and $\Delta_R \approx 25$. The reformation delay improves by about 4%, but the component of the message delay more than doubles. Finally, if $\Delta_C = \Delta_R = \Delta_T$, as in the initial system, $\Delta_T + \Delta_C \approx 48$ and $\Delta_R \approx 24$. The message delay is about 3.5 times as large as it is in the system with independent selections, while the reformation delay improves by less than 8%. This example shows the importance of separating the selection of the delay.

6 Conclusion

We have shown that the guaranteed delivery delay in T-RMP can be reduced by combining the retry counters used to recover the token passing message and the missing source messages, and by using different time intervals to pass the token, Δ_T, commit messages, Δ_C, and to enter the reformation process, Δ_R. It is instructive to determine the reductions using reasonable numbers.

In original system $\Delta_T = \Delta_C = \Delta_R$, and the same number of retries n_r is allowed to recover the token passing message and the source messages. The maximum time from the reception of a source messages until it committed by all of the receivers is $\Delta_S = \Delta_T + \Delta_C$. In the initial system, $\Delta_T = (2n_r + .5)\Delta_N$, where n_r is the number of retries used to recover a missing message, and Δ_N is the nominal round trip network delay that we use to retransmit message recovery requests. The factor of 2 results from the two separate message recovery processes, and the factor .5 is the time that a receiver waits for the token passing message before initiating the recovery process. When the retry count for the two recovery processes are combined and $P_L \ll 1$, the total number of retries is limited to $n_t = n_r + 1$, so that $\Delta_T = (n_r + 1.5)\Delta_N$.

In typical ARQ protocols $n_r = 3$. Therefore, Δ_S in the combined system is $\frac{4.5}{7.5} = .6$ as large as in the original system. If the nominal round trip delay is one second, the maximum source delay is 13 seconds in the original system and 9 seconds in the combined system.

In the figures in section 5 all of the delays are normalized with respect to μ_s. If the selection of $n_r = 3$ and a nominal network delay of 1 second results in a probability of

erroneous reformation $= 10^{-10}$, then, from the example at the end of section 5, $\Delta_T = \Delta_C = \Delta_R = 24\mu_s$. If we select the three periods independently, and allow 10^{-4} of the messages to arrive at some receivers after they have been committed by other receivers, then we can set $\Delta_T = 3\mu_S$, $\Delta_C = 11\mu_S$, and $\Delta_R = 28\mu_S$, and maintain $P_r = 10^{-10}$. This reduces Δ_S from 48 to 14, and the maximum source delay from 9 seconds to 2.625 seconds.

The two protocol modifications that we have studied provide a reduction in the delivery delay, in this example, of nearly 80%. It is worth noting that the guarantee is approaching the nominal network delay, so it is unlikely that further protocol modification will provide large improvements. In order to provide stronger delay guarantees we have to increase the number of retries in order to decrease the nominal network delay toward μ_s, or improve the operation of the network to reduce the actual network delay.

References

[1] A. N. Netravali, W. Roome, K. K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol," IEEE Trans. Comm., vol. 38, no. 11, Nov. 1990, pp. 2010-2024.

[2] J-M. Chang, N. F. Maxemchuk, "Reliable Broadcast Protocols," ACM Transactions on Computer Systems, Vol. 2, No. 3, Aug. '84, pp. 251-273.

[3] N. F. Maxemchuk, D. Shur, "An Internet Multicast System for the Stock Market," ACM TOCS, Aug. 2001.

[4] J-M. Chang, "Simplifying Distributed Database Systems Design by Using a Broadcast Network," Proc SIGMOD '84, pp 223-233, June 1984.

[5] B. Whetten, G. Taskale, "An overview of reliable multicast transport protocol II," IEEE Network Mag., Jan/Feb 2000, pp 37-47.

[6] N. F. Maxemchuk, J-M. Chang, "Analysis of the Messages Transmitted in a Broadcast Protocol," Proc ICC '84, pp 1263-1267, May, 1984.

[7] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," IEEE Trans. on Communications, Vol. 39, pp. 1482-1493, Oct. 1991.

[8] D. L. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks," IEEE/ACM Trans. on Networking, Vol. 3, No. 3, pp. 245-254, June 1995.

[9] A. Ciuffoletti, "Uniform timing of a multi-cast service," Proc. of IEEE Conf. on Distributed Computing Systems, May 31- June 4, 1999. pp. 478-486.

[10] L. Kleinrock, **Queueing Systems -- Volume 1: Theory,** John Wiley & Sons, 1975.