Using Adaptive Timeouts to Achieve At-Most-Once Message Delivery *

Soma Chaudhuri,[1] Brian A. Coan,[2] and Jennifer L. Welch[3]

[1]Iowa State University, Ames, IA 50011, USA
[2]Bellcore, 445 South Street, Morristown, NJ 07960, USA
[3]Texas A&M University, College Station, TX 77843, USA

April 18, 1995

Abstract

We extend the at-most-once message delivery algorithm of Liskov, Shrira, and Wroclawski to adapt dynamically to changes in message transmission time and degree of clock synchronization. The performance of their algorithm depends on its being supplied with a good estimate of the maximum message lifetime—the sum of the message delivery time and the difference in processor clock values between sender and recipient. We present two algorithms that are suitable for use in a system where the message lifetime is unknown or may change. Our extensions allow the automatic and continuous determination of a suitable value for the maximum lifetime. We prove that whenever the actual message lifetime is bounded, then our adaptive algorithms converge to an accurate estimate of its true value. Our two algorithms differ in the behavior they require from the network and achieve different performance levels. Our formal statement of convergence is expressed in terms of the number of messages received, rather than time elapsed. We show that this formulation is necessary by proving that no method for estimating the lifetime can achieve convergence in a bounded amount of time.

Key words: At-most-once message delivery – Communication algorithms – Synchronized clocks – Adaptive algorithms

*This work was done while the first and third authors were at the University of North Carolina at Chapel Hill, supported in part by NSF grant CCR-9010730, an IBM Faculty Development Award and NSF PYI Award CCR-9158478. A preliminary version of this paper appeared in the Proceedings of the Fifth International Workshop on Distributed Algorithms, pp 151–166, 1991.

1 Introduction

Delivering messages from one computer to another over a communication network is necessary in order to make distributed systems usable. In an ideal world, a message delivery algorithm would deliver exactly one copy of each message sent. This ideal behavior is generally impossible to implement in the presence of network and processor misbehavior. In practice, it is common for a message delivery protocol to make an *at-most-once* guarantee. In the at-most-once message delivery problem, each of a number of senders wishes to send messages to a receiver. The protocol should never deliver multiple copies of a sent message, even in the presence of failures. Additionally, it should make a best effort to deliver one copy and fail at this only in the presence of network or processor misbehavior. Such at-most-once messages can be combined with other low-level communication primitives, including unreliable datagrams, to provide higher-level communication primitives. An apt example is connection management—at-most-once messages are used for connection set-up, and then datagrams are used for the data stream. (Cf. discussion in [7].)

Liskov et al. [7] describe an algorithm for at-most-once message delivery that takes advantage of synchronized clocks. It uses an estimate of the maximum message lifetime—the sum of the maximum message delay and the maximum deviation of the sender's clock from the recipient's clock—to allow the recipient to "forget" all state information about senders that have been inactive for sufficiently long. This state information is stored in a data structure called a connection table. Because of the "forgetting," the connection table may be substantially smaller than would otherwise be required. Crash resiliency of the server can be achieved without storing the connection table in stable storage. Instead a loose upper bound suffices, thus reducing the number of disk writes required.

The Liskov et al. algorithm must be supplied with an estimated upper bound* on the message lifetime. Mistakes in making this estimation do not compromise correctness, but they can damage performance. The cost of having too small an estimated bound is that many messages can be "lost," i.e., erroneously discarded as duplicates. The cost of having too large an estimated bound is that connection-specific information is retained for longer than necessary.

We present two extensions to the Liskov et al. algorithm for at-most-once message delivery. Each algorithm uses a different method for dynamically determining an estimate of the message lifetime. Our idea of dynamically changing the estimate of the lifetime is inspired by prior work on adaptively changing timeout intervals [3, 10]. We demonstrate that both our methods converge to an appropriate value whenever the actual message lifetime in the system is sufficiently well-behaved. Specifically, we demonstrate that our estimate is not too high by proving that it is no more than twice the true value, and we demonstrate that our estimate is not too low by proving that, as the length of an execution tends to infinity, the fraction of messages that are erroneously rejected because of the estimate tends to some

*As Liskov et al. discuss, the clients can be partitioned into different classes (e.g., local and remote) with different estimates of the maximum message lifetime maintained independently for each class. We will not discuss this point any further in this paper.

suitable value. Our formal statement of the rate at which the message-lifetime estimate converges to the true value is expressed in terms of the number of messages received, rather than time elapsed. We conclude the paper by showing that this formulation is necessary. Specifically, we prove that no method for estimating the lifetime can achieve convergence in a bounded amount of time.

There are three reasons why it is useful for a distributed algorithm to have the ability to continuously adapt to the current prevailing message lifetime. First, it simplifies the job of configuring a system. There is no need to determine the lifetime and there is no need to install this parameter in the code for the algorithm. Second, in many systems the load, and hence the lifetime, varies in an approximately periodic way. For instance, days may be relatively busy and nights relatively idle. In such systems using an adaptive algorithm makes it possible to automatically adjust the estimate of the lifetime accordingly. Third, it is reasonable to expect that over the long term there may well be a change in the message lifetime in a system. Failure to use an adaptive algorithm could result in sudden and unexpected system collapse should the actual message lifetime drift beyond the expected range.

Our two algorithms, called the unlimited horizon algorithm and the limited horizon algorithm, differ in the following three respects: the requirements for the system to be considered well-behaved, the convergence properties that they ensure in well-behaved executions, and the extent to which the current behavior of the algorithm depends on history.

For the unlimited horizon algorithm, an execution is considered well-behaved if there is some time after which there is an upper bound on the lifetime of any message. This algorithm ensures that in well-behaved executions, as the length of the execution tends to infinity, the fraction of messages that are erroneously rejected (due to the lifetime estimate) tends to 0. To achieve this convergence property, the behavior of this algorithm depends heavily on history: the longer the execution, the more reluctant the server is to reduce its estimate of message lifetime.

The limited horizon algorithm eliminates this undesirable dependence on history and has a more lenient notion of well-behaved executions. For this algorithm an execution is considered well-behaved if there is some time after which there is an upper bound that holds on the lifetime of *most* messages. (A more precise definition is in the body of the paper.) This algorithm ensures that in well-behaved executions, as the length of the execution tends to infinity, the ratio of messages that are erroneously rejected (due to the lifetime estimate) to messages that are accepted tends to δ, for (almost) any fixed δ. Note that this is a weaker convergence property than that of the unlimited horizon algorithm. We believe that its reduced sensitivity to history and its ability to ignore the occasional very late message contribute to the practical utility of the limited horizon algorithm.

Our algorithms, like the Liskov et al. algorithm on which they are based, are examples of the type of algorithm advocated by Liskov [6] that takes advantage of practical, yet probabilistic network clock synchronization algorithms, such as that of Mills [8]. Because of its great efficiency, the Mills algorithm makes it completely feasible to write distributed algorithms that use approximately synchronized processor clocks. However, because there is a small possibility that processor clocks will have a larger deviation than expected, it is

important to avoid distributed algorithms whose correctness relies on synchronized clocks. Rather, the clocks should be used as a "hint," which is important for performance, but is not needed for correct operation. In the at-most-once protocols, should the clocks deviate more than expected, the upper bound on message lifetime may be violated, causing lost messages (a performance problem), but the at-most-once property (correctness) is not violated. Other examples of algorithms that use time as a hint include the orphan elimination algorithm of Herlihy and McKendry [2] and the Kerberos authentication protocol [9].

Time-based algorithms may become more important as network throughput increases. In the connection management application, the sequence of control messages for connection set-up (for a specific pair of nodes) can be viewed as the sequence of messages for at-most-once message delivery. Unlike in TCP-IP (the Internet transmission control protocol) [1], in a time-based algorithm no handshake is required between the sender and recipient to initiate the connection. As the transmission speed of a network increases, the ratio between the time it takes to do a handshake—which is bounded below by the propagation time of light—and the time it takes for each packet to enter (or leave) the network increases. Thus, the elimination of the handshake becomes proportionately more valuable as the speed of the network goes up. It is also particularly advantageous when a connection is used for very few messages, as is often the case in client-server interactions such as remote-procedure calls.

2 Problem statement and assumptions

The system in which an at-most-once message delivery algorithm operates consists of one server process and a collection of client processes. These processes run on a collection of processors. They interact by sending messages over a communication network. The network can lose, duplicate and reorder messages, delivering zero, one, or many copies of each message that is sent.

Each client has a sequence of unique messages to be sent to the server. The specific behavior of each client is to progress through its message sequence one message at a time, sending a single copy of the current message. Each message is sent at a different time and the times are monotonically increasing. Before proceeding on to send the next message in the sequence, the client waits until either it has learned of the acceptance of the current message (due to an acknowledgment received from the server) or it has given up on the acceptance of the current message. A message on which the client has given up may or may not eventually be accepted by the server, but it will not be accepted out of order. Clients do not retransmit (possibly lost) messages. When the server receives a message from a client it either accepts or rejects it. Absolute requirements on the server are that it never accept a second copy of a message that it has already accepted (a *duplicate* message) and that it never accept a message that was sent earlier than the last accepted message from the same client. The server may on occasion reject messages that are not duplicates and are not out-of-order. To ensure that progress is made, it is desirable for the server to accept as many non-duplicate, in-order messages as practicable.

We restrict our attention to algorithms that conform to a framework suggested by the Liskov et al. algorithm. We now describe that framework. Throughout this paper, all times referred to are times on the server's clock (unless otherwise explicitly stated).

Every message sent by a client is tagged with the current value of the client's clock, the "timestamp." The server keeps a cache CT ("connection table") with an entry for each client. $CT[i]$ holds the timestamp of the most recent message from client i that has been accepted by the server. If the server does not hear from client i for a sufficiently long time, then the entry is replaced with *nil* ("garbage collected"). The server also keeps a variable *upper*, which is an upper bound on the largest timestamp of any entry that has ever been garbage collected from the connection table.

We now describe what happens when the server receives a message with timestamp t from client i. If t is greater than the (non-*nil*) entry in CT for i, then this message is not a duplicate and is not out-of-order and it is accepted. If t is less than or equal to the (non-*nil*) entry in CT for i, then this message is a duplicate or is out-of-order and it is rejected. If the entry in CT for i is *nil*, then t is compared against *upper*. If t is larger, then this message is not a duplicate and is not out-of-order, since *upper* is an upper bound on the largest garbage-collected entries in CT. The message is accepted, and t is stored in $CT[i]$. If t is less than or equal to *upper*, then this message is either (1) a duplicate or out-of-order message or (2) a late-arriving message that is neither a duplicate nor out-of-order. Because these two situations are indistinguishable, the message is rejected.

Every so often the connection table is garbage collected based on an estimate of the maximum message lifetime (the sum of the message delivery time and the uncertainty in the clock synchronization). For each entry in CT, the difference between that entry and the current clock value at the server is calculated. If this difference is greater than the estimate of maximum message lifetime, then the entry is garbage collected. When an entry is garbage collected, it is replaced by *nil* and *upper* is updated if necessary. The updating of *upper* is done to maintain the invariant that *upper* is at least as big as the largest entry that has ever been garbage collected from the connection table.

The Liskov et al. algorithm uses a fixed value for the lifetime estimate. In this paper we develop two methods for automatically and continually estimating the lifetime. We use these methods in conjunction with the basic framework of the Liskov et al. algorithm. Throughout the remainder of this paper we use the term *standard-form algorithm* to refer to an algorithm that uses the basic framework of the Liskov et al. algorithm (e.g., a connection table, *upper*, and periodic garbage collection), but with estimates of the maximum lifetime that are computed as the execution proceeds.

We now define some terms that we will be using to discuss our algorithms.

The *lifetime* of a message received by the server in an execution is the difference between the time on the server's clock when the server receives the message and the time on the client's clock when it sends the message. The lifetime has two components, the message delay plus the difference (either positive or negative) in clock skew between the server's clock and the client's clock. The server can calculate the lifetime of a message if the sender appends its current clock time to the message. In our algorithms, the client always does this.

A message m from client i is *out-of-order* in an execution if it arrives at the server after message m' from client i arrives at the server, where i sent m before it sent m'. A message is *lost* if it is not a duplicate, is not out-of-order, and, when it arrives at the server, it is rejected. A message is *summary-rejected* if it is rejected by comparing its timestamp with *upper*. A message is *table-rejected* if it is rejected based on a comparison with a non-*nil* entry in CT. Note that no table-rejected messages are lost, but that some summary-rejected messages may be lost.

Given a time t in an execution, let A_t be the number of messages accepted by the server up to time t, and let L_t be the number of messages lost by the server up to time t.

For time t and positive integer X, an execution is (t, X, S, H)-*bounded* provided that in every consecutive group of S messages arriving after time t, at most H messages have lifetime greater than X. If $H = 0$, then the parameter S gives no information, and we use the simpler notation (t, X)-*bounded*.

We would like to guarantee the following conditions for fixed integers S and H and real number $\delta \geq 0$.

- **Condition 1:** No duplicate or out-of-order messages are accepted by the server in any execution.
- **Condition 2:** In any execution that is (t, X, S, H)-bounded for some t and X, there is some time $t' > t$ after which the server's estimate of the lifetime is always less than $2X$.
- **Condition 3:** In any execution that is (t, X, S, H)-bounded for some t and X, the limit of L_u/A_u is at most δ as u tends to infinity.

The first condition is the basic safety condition for any at-most-once message delivery algorithm: specifically, it is that no duplicate or out-of-order messages ever be accepted by the server. The next two conditions ensure that the performance of our algorithm is adequate. The second condition implies that the algorithm does not use an excessive amount of memory in the connection table or store old connection information for an excessively long period of time. The third condition requires that there be an adequate amount of progress (i.e., number of non-duplicate, non-out-of-order messages accepted).

In order to achieve these conditions, we must make five assumptions. The first assumption is needed because our conditions are inherently about executions in which an infinite number of messages are sent and received.

- **Assumption 1:** The server receives an infinite number of non-duplicate messages in any infinite execution.

The second assumption is needed to prevent an adversary from swamping an algorithm with messages during some brief period when the bound on message lifetime is set too low. Without assuming an upper bound on the rate with which messages arrive at the server, the most we can guarantee is this: for the chosen δ, there is an infinite number of times u in the execution where $L_u/A_u \leq \delta$.

- **Assumption 2:** There is an integer R such that for any times t_1 and t_2 in any execution, $t_1 < t_2$, the server receives at most $(t_2 - t_1) \cdot R$ messages in the interval $[t_1, t_2]$.

Since, even when the execution is well-behaved, at least H messages (the ones making up the transient spike) can be summary-rejected out of every S messages received, we cannot hope to achieve our goals unless the desired value of δ is not too small.

- **Assumption 3:** $\delta \geq H/(S - H)$.

Because we use timestamps to deduce the order in which messages are sent, we require that a client never make a negative adjustment to its clock. It is known [4, 5] that this assumption does not preclude the use of clock synchronization algorithms.

- **Assumption 4:** A client never sets its clock backwards.

Our last assumption is that message lifetimes are always positive. The underlying network layer can enforce this by artificially delaying messages if necessary. However, we believe that such delaying would seldom be necessary, due to clocks being closely synchronized and the time overhead to send a message. As a technical convenience, we select our time units so that one clock tick of a correctly running clock is exactly one unit of time. Because of this selection Assumption 5 actually gives us the property that the lifetime of every message is greater than or equal to 1.

- **Assumption 5:** The lifetime of every message received by the server is positive.

3 The algorithms

We begin this section with a general definition of standard-form algorithms for at-most-once message delivery; these are algorithms that follow the basic framework given by Liskov et al. [7]. We give the code for a general standard-form algorithm in Section 3.1 and we prove that any standard-form algorithm satisfies Condition 1. Then we give our two specific standard-form algorithms—the unlimited horizon algorithm and the limited horizon algorithm—and prove that each of them satisfies Conditions 2 and 3.

The unlimited horizon algorithm achieves a ratio of lost to accepted messages that approaches 0 (i.e., $\delta = 0$), but it requires that there be a time after which every message received, without exception, has a fixed upper bound on its lifetime. Also, the longer the system has been running, the longer it takes the estimated lifetime to be reduced once the observed lifetimes become small.

The limited horizon algorithm only achieves a ratio of lost to accepted messages that approaches a constant δ, where the choice of δ is subject to the constraint given in Assumption

3 and to the additional constraint that $\delta = 1/p$ for some positive integer p. However, it can tolerate transient spikes in the lifetimes of messages received, and the convergence behavior of the estimated lifetime does not depend on the entire history.

The times at which garbage collections can be done affect the method that can be used to tolerate server crashes, as discussed in Liskov et al. [7]. In the unlimited horizon algorithm, garbage collection can be done at arbitrary times, as long as there is an upper bound on the time between garbage collections. Thus either of the two crash recovery methods presented by Liskov et al. can be used. In the limited horizon algorithm, garbage collection must be done exactly when a certain number of messages have been received since the last garbage collection. Since there is no fixed upper bound on the elapsed time between two garbage collections, only one of the two crash recovery methods of Liskov et al. can be used with this algorithm.

3.1 Standard-form algorithms

A *standard form algorithm* has as its skeleton the code given in Fig. 1. The only additions that are permitted to the skeleton are arbitrary (terminating) code for the computation of the variable *ltime-est* ("lifetime-estimate") and an arbitrary collection of auxiliary state variables to be used in the the computation of *ltime-est*. The role of the auxiliary state variables is to permit the computation of *ltime-est* to be based on any arbitrary property of the prior execution. Code to update the auxiliary state variables can be added anywhere in the skeleton. The only permitted dataflow back from the auxiliary variables to the skeleton code is through the variable *ltime-est.*

Each client is assumed to timestamp each message it sends with the current value of its clock. The variable CT would actually be implemented in a more space-efficient way than a static array, say with a hash table or a balanced tree.

Theorem 2 states that standard-form algorithms never accept duplicate or out-of-order messages. Its proof makes absolutely no assumptions about the way that the variable *ltime-est* is computed. It is proved using a collection of auxiliary functions lim_i whose behavior is characterized in Lemma 1. For any state of the algorithm (i.e., after the execution of any line of code), lim_i is defined to be $CT[i]$ if $CT[i]$ is not *nil* and *upper* otherwise.

Lemma 1 *In any execution of a standard-form algorithm, for any i, the sequence of values taken on by lim_i is non-decreasing.*

Proof: Consider the three places in the code that could potentially change the value of lim_i.

Case 1: ($CT[i]$ is set to the timestamp of an accepted message.) The precondition for accepting this message is exactly that its timestamp be at least as big as the current value of lim_i.

State variables:

clock: integer, current value of the clock
$CT[1..n]$: array of integer, initially all entries are *nil*
ltime-est: integer
upper: integer, initially 0

Transitions:

- Receive message m from client i with timestamp *stamp*:

  if $((CT[i] \neq nil)$ and $(stamp > CT[i]))$ or $((CT[i] = nil)$ and $(stamp > upper))$ then
    accept m and send acknowledgement
    $CT[i] := stamp$
  else
    reject m

- Garbage collect CT:

  ltime-est := any function of the state variables
  for $i = 1$ to n do
    if $(CT[i] \neq nil)$ and $(CT[i] \leq clock - ltime\text{-}est)$ then
      $upper := \max(upper, CT[i])$
      $CT[i] := nil$

Figure 1: The skeleton of a standard-form algorithm

Case 2: (*upper* is changed.) This only happens when $CT[i]$ is not *nil* and so does not change the value of lim_i.

Case 3: ($CT[i]$ is set to *nil*.) This happens in only one place in the code. In the previous statement *upper* is set equal to the maximum of $CT[i]$ and something else. Thus lim_i is not decreased. ■

Theorem 2 *A standard-form algorithm never accepts the same message more than once and never accepts an out-of-order message in any execution.*

Proof: Suppose message (m, u, i), denoting a message with content m and timestamp u from client i, arrives at the server and is accepted, and subsequently message (m', u', i) arrives at the server with $u' \leq u$. (If $u' = u$, then the second message is a duplicate, otherwise it is out-of-order.) When (m, u, i) arrives, $CT[i]$ is set equal to u and so lim_i is u. Consider the time t when (m', u', i) arrives. By Lemma 1, at time t, $lim_i \geq u$ and thus the precondition for accepting (m', u', i) is not satisfied and the message is rejected. ■

3.2 The unlimited horizon algorithm

In Fig. 2 we describe the unlimited horizon algorithm for the server, assuming n clients. The unlimited horizon algorithm is obtained by making some additions to the skeleton standard-form algorithm. Each line added to the standard-form code is marked with an asterisk.

State variables:

$clock$: integer, current value of the clock
$CT[1..n]$: array of integer, initially all entries are nil
$ltime\text{-}est$: powers of 2, initially 1
$upper$: integer, initially 0
* p: integer, initially 1
* $numacc$: integer, initially 0
* $numrej$: integer, initially 0
* $maxltime$: integer, initially 0
* $ltimes$: multiset of integer, initially empty

Transitions:

• Receive message m from client i with timestamp $stamp$:
* $ltimes := ltimes \cup \{clock - stamp\}$
if $((CT[i] \neq nil)$ and $(stamp > CT[i]))$ or $((CT[i] = nil)$ and $(stamp > upper))$ then
accept m and send acknowledgement
$CT[i] := stamp$
* $numacc := numacc + 1$
else
reject m
* if $(CT[i] = nil)$ and $(stamp \leq upper)$ then $numrej := numrej + 1$

• Garbage collect CT:
* $maxltime :=$ the largest element in $(ltimes \cup \{0\})$
* if $maxltime > ltime\text{-}est$ then
* $ltime\text{-}est := 2^j$, where $j = \min\{k : 2^k \geq maxltime\}$
* else if $(numacc > p \cdot numrej)$ and $(ltime\text{-}est > maxltime > 0)$ then
* $ltime\text{-}est := 2^j$, where $j = \min\{k : 2^k \geq maxltime\}$
* $numacc := numacc - p \cdot numrej$
* $numrej := 0$
* $p := p + 1$
* $ltimes :=$ empty set
for $i = 1$ to n do
if $(CT[i] \neq nil)$ and $(CT[i] \leq clock - ltime\text{-}est)$ then
$upper := \max(upper, CT[i])$
$CT[i] := nil$

Figure 2: The unlimited horizon algorithm

The server has two transitions. Whenever a message arrives, the transition includes the lifetime of the current message in the multiset *ltimes*.[†] It increments *numacc* if the message is accepted, and increments *numrej* if the message is summary-rejected.

The server's transition for garbage collecting the connection table can occur at arbitrary times as long as there is an upper bound G on the amount of (server's clock) time elapsing between two consecutive garbage collections. Within each garbage-collection period, the server keeps a multiset of the lifetimes of all the messages received. At the end of each garbage-collection period, the server has the option of changing its current maximum lifetime estimate. The change can be either an increase or a decrease.

If the maximum lifetime observed is greater than the current estimate, then the current estimate is increased to be the smallest power of 2 at least as large as the maximum lifetime observed. The allowable values for the lifetime estimates are restricted to powers of 2 in order to reduce the number of step increases needed, and thus the number of messages lost, until the maximum estimate is achieved, once the execution is well-behaved.

If the maximum lifetime observed is smaller than the current estimate, then the current estimate may be decreased to be the smallest power of 2 at least as large as the maximum lifetime observed. The decrease happens if the number of messages accepted is "sufficiently larger" than the number of messages summary-rejected. What we mean by "sufficiently larger" depends on the number of times the estimate has been decreased thus far. The p-th time the estimate is decreased, the number of messages accepted must be larger than p times the number of messages summary-rejected. (This condition is implemented with the variables *numacc*, *numrej*, and p.) Once the lifetime estimate has been adjusted, it is used to garbage collect the connection table in the usual way.

The unlimited horizon algorithm satisfies the three correctness conditions with $H = 0$ and $\delta = 0$. Condition 1 follows from Theorem 2 since the unlimited horizon algorithm is a specific type of standard-form algorithm. Conditions 2 and 3 follow from Theorem 4 and Theorem 6 respectively. Theorem 4 states that, once the execution is well-behaved, i.e., there is an upper bound on the maximum lifetime, the lifetime estimate will eventually converge to a value less than twice the upper bound. Theorem 6 states that in any well-behaved execution, the ratio of lost to accepted messages tends to 0. The proofs of these two theorems rely on Lemma 3, which gives an upper bound on how long it takes after the estimate stabilizes until no message is summary-rejected. The proof of Theorem 6 also makes use of Lemma 5, which gives an upper bound on the number of messages that can be summary-rejected in certain intervals of time.

Lemma 3 *Let t_1 be any garbage-collection time in any execution of the unlimited horizon algorithm. Let B be the new value of ltime-est at time t_1. If t_2 is a garbage-collection time such that $t_1 + B < t_2$, and ltime-est is non-increasing in the closed interval $[t_1, t_2]$, then no message is summary-rejected in $[t_1 + B, t_2]$.*

[†] The multiset is used in order to parallel the code of the limited horizon algorithm, presented next; an efficient way to implement the unlimited horizon algorithm is only to keep track of the maximum lifetime observed since the last garbage collection.

Proof: Consider a particular message that arrives at time t in the interval $[t_1+B, t_2]$. Let s be the value of *ltime-est* at time t. Then, the timestamp of the message is at least $t-s$, since *ltime-est* is not increased in this interval. To show that this message is not summary-rejected, it is sufficient to show that the value of *upper* at time t is at most $t-s$.

Suppose there is no change in the value of *upper* between times t_1 and t. The value of *upper* at time t_1 is at most t_1. Thus, the value of *upper* at time t is still at most t_1. Since *ltime-est* is non-increasing in $[t_1, t_2]$, this implies that B is an upper bound on the value of *ltime-est* in this interval and hence on s. Because $t \geq t_1 + B$ and $s \leq B$, it follows that $t - s \geq t_1$. Therefore, the value of *upper* at time t is at most $t-s$.

Suppose there is a change in the value of *upper* between times t_1 and t. (All changes to *upper* are made during garbage collection.) Let t' be the time of the last change to *upper* in the interval $[t_1, t]$ and let s' be the new value of *ltime-est* at time t'. At time t', *upper* is set to some value that is at most $t' - s'$, since *upper* is the largest timestamp garbage-collected and $t' - s'$ is the threshold for garbage collection. Thus at time t, *upper* has not changed and is still at most $t' - s'$, which is less than $t - s'$ (since t' comes before t), which is at most $t - s$ (since *ltime-est* is non-increasing). ■

Theorem 4 *Consider any execution of the unlimited horizon algorithm that is (t, X)-bounded for some t and X. Then there exists some real time $t' > t$ such that the value of the variable ltime-est at every time after t' is less than $2X$.*

Proof: Suppose the value of *ltime-est* is always less than $2X$ after time t. Then, we are done. Otherwise, let $t_1 > t$ be a point in time when the value of *ltime-est* is at least $2X$.

We show that $numacc > p \cdot numrej$ at some point t'' after t_1. Let s be the value of *ltime-est* at time t_1. Suppose the value of *ltime-est* decreases at some point after t_1. This means that $numacc > p \cdot numrej$ at that time. Otherwise, suppose *ltime-est* is never decreased. Since the execution is (t, X)-bounded, every message received after t_1 has lifetime at most X, and thus *ltime-est* is never increased. Then Lemma 3 applies to any interval starting at t_1, with $B = s$, and no message that arrives after $t_1 + s$ is summary-rejected. Therefore, $numacc > p \cdot numrej$ at some point $t'' \geq t_1$.

Now, at every garbage collection after t'', the server sets *ltime-est* to be the smallest power of 2 at least as large as the largest message lifetime it observed since the last garbage collection. By the (t, X)-bounded assumption, the largest lifetime observed is X, and thus *ltime-est* is always set to less than $2X$. So, starting at the time t' of the first garbage collection after t'', *ltime-est* is less than $2X$. ■

Plotting the value of *ltime-est* versus time produces a step function. Let $u_0 = 0$ and let u_i identify the point in time when *ltime-est* is decreased for the i-th time. At this time, *numrej* and *numacc* are reset and p is incremented. Define the i-th *interval* I_i to be the half open interval $[u_{i-1}, u_i)$, $i \geq 1$. Consider the behavior of *ltime-est* in I_i. I_i consists of a sequence of one or more maximal *sub-intervals* which have the property that *ltime-est* is constant within

each sub-interval. Each sub-interval consists of a sequence of consecutive garbage-collection periods.

Let r_i be the number of messages summary-rejected in I_i. Notice that this is equal to the value of *numrej* at the end of I_i just before it is reset to 0. The variable p is set to the value i at the beginning of I_i. Let a_i be the amount subtracted from *numacc* at the end of I_i, namely $i \cdot r_i$.

Lemma 5 *Consider any execution of the unlimited horizon algorithm that is (t, X)-bounded for some t and X. Then there exists a constant Z (depending on X) such that for all i with I_i beginning after t, $r_i \leq Z$.*

Proof: Consider any i with I_i beginning after t. I_i consists of one or more sub-intervals, where the value of *ltime-est* in each sub-interval is larger than its value in the preceding sub-interval. Each time *ltime-est* is increased, *ltime-est* is at least doubled, and the maximum value that *ltime-est* can reach is $2X$. Thus there are at most $\lceil \log X \rceil + 1$ sub-intervals in I_i.

Now, since *ltime-est* is constant in each sub-interval, the sub-intervals satisfy the condition of Lemma 3. Therefore, we can apply Lemma 3, with $B = 2X$, to each sub-interval (note that the maximum value of *ltime-est* in each sub-interval is at most $2X$). So, it follows that, in each sub-interval, messages are only summary-rejected in the last garbage-collection period before *ltime-est* is increased, and in the first $2X$ time units after the beginning of the sub-interval. Therefore, the messages can be summary-rejected for at most $2X + G$ time in every sub-interval, where G is the maximum length of a garbage-collection period. Thus the maximum number of messages that can be summary-rejected in I_i is $Z = (\lceil \log X \rceil + 1) \cdot (2X + G) \cdot R$, where R is the maximum rate at which the server receives messages. ■

Theorem 6 *Consider any execution of the unlimited horizon algorithm that is (t, X)-bounded for some t and X. Then the limit of L_u/A_u as u tends to infinity is 0.*

Proof: We must show that for any ϵ, there exists a time t_ϵ such that for all $u \geq t_\epsilon$, $L_u/A_u \leq \epsilon$.

Suppose there exists a time $t_f \geq t$ after which *ltime-est* never changes. Lemma 3 gives the result.

Now suppose that *ltime-est* changes infinitely often. Clearly *ltime-est* is always at least 1. By Theorem 4, there exists a $t' > t$ such that *ltime-est* never exceeds $2X$ after time t'. Thus, *ltime-est* increases infinitely often and decreases infinitely often.

Choose q to be a positive integer such that $\epsilon \geq 1/q$. We show that there exists t_ϵ such that $L_u \cdot q \leq A_u$ for all $u \geq t_\epsilon$.

Pick any time u such that u is in I_i for some $i > q$ and I_i begins after t. Then $L_u \leq r_1 + \cdots + r_{i-1} + r_i$ and $A_u \geq a_1 + \cdots + a_{i-1}$. By definition, $a_j = j \cdot r_j$ for all j. Thus

$$\begin{aligned} a_1 + \cdots + a_{i-1} &= 1 \cdot r_1 + \cdots + (i-1) \cdot r_{i-1} \\ &= q \cdot (r_1 + \cdots + r_{i-1}) - Q + R_{i-1}, \end{aligned}$$

where $Q = (q-1)\cdot r_1 + (q-2)\cdot r_2 + \cdots + r_{q-1}$ and $R_{i-1} = r_{q+1} + 2\cdot r_{q+2} + \cdots + (i-1-q)\cdot r_{i-1}$. Thus,

$$\begin{aligned} L_u \cdot q &\leq q\cdot(r_1 + \cdots + r_{i-1}) + q\cdot Z, && \text{since } r_i \leq Z \text{ by Lemma 5} \\ &= a_1 + \cdots + a_{i-1} + Q - R_{i-1} + q\cdot Z \\ &\leq A_u + Q - R_{i-1} + q\cdot Z. \end{aligned}$$

We have two cases. If there exists m such that $r_j = 0$ for all $j > m$, then L_u is bounded above by some constant for all u. Therefore, $L_u \cdot q$ is bounded above by some constant for all u. Since A_u approaches infinity as u increases, there is some t_ϵ such that $L_u \cdot q \leq A_u$ for all $u \geq t_\epsilon$.

If, on the other hand, there is an infinite number of non-zero r_j's, then R_{i-1} approaches infinity as i increases. Since Q and $q\cdot Z$ are constants, there is some i_0 such that for all $i \geq i_0$, $R_{i-1} \geq Q + q\cdot Z$. Let t_ϵ be the beginning of I_{i_0}. For all times $u \geq t_\epsilon$, $L_u \cdot q \leq A_u$. ■

3.3 The limited horizon algorithm

In Fig. 3 we describe the limited horizon algorithm for the server, assuming n clients. The limited horizon algorithm, like the unlimited horizon algorithm, is obtained by making some additions to the skeleton standard-form algorithm. Each addition is marked with an asterisk.

The limited horizon algorithm achieves the three correctness conditions for any fixed S, H, and δ, where the choice of δ is subject to the constraint given in Assumption 3 and to the additional constraint that $\delta = 1/p$ for some positive integer p. The three constants S, H, and p are embedded in the server's code.

The server has two transitions. The behavior when a message arrives in the limited horizon algorithm is identical to the behavior in the unlimited horizon algorithm.

The transition for garbage collecting the connection table occurs after every S-th message arrival. The server keeps a multiset of the lifetimes of all the messages received during the current garbage-collection period. When S messages have been received (and either accepted or summary-rejected), then the server tries to update the lifetime estimate and garbage collects the connection table. In determining how to adjust the estimate, the server compares the $(H+1)$-st largest element in the multiset of recorded lifetimes against the current lifetime estimate. Based on this comparison, the estimate is updated in much the same way used in the unlimited horizon algorithm.

There are two main ways in which the update procedure used in the limited horizon algorithm differs from that used in the unlimited horizon algorithm. First, the variable p has a fixed value throughout. As will be shown, this allows the ratio of lost to accepted messages to converge to $1/p$, as long as p is at most $(S-H)/H$. No advantage regarding the ratio can be gained with a larger p since H out of every S messages might be summary-rejected simply due to transient spikes in the lifetimes. Second, whenever the lifetime estimate is decreased, *numrej* is set to 0 and *numacc* is decreased by p times *numrej* plus 1. The plus 1 is important for achieving the desired ratio—it allows us to build up a reserve of

State variables:

$clock$: integer, current value of the clock
$CT[1..n]$: array of integer, initially all entries are nil
ltime-est: powers of 2, initially 1
$upper$: integer, initially 0
* p: integer, initially some predetermined value at most $(S-H)/H$
* $numacc$: integer, initially 0
* $numrej$: integer, initially 0
* $maxltime$: integer, initially 0
* $ltimes$: multiset of integer, initially empty

Transitions:

• Receive message m from client i with timestamp $stamp$:
* $ltimes := ltimes \cup \{clock - stamp\}$
if $((CT[i] \neq nil)$ and $(stamp > CT[i]))$ or $((CT[i] = nil)$ and $(stamp > upper))$ then
accept m and send acknowledgement
$CT[i] := stamp$
* $numacc := numacc + 1$
else
reject m
* if $(CT[i] = nil)$ and $(stamp \leq upper)$ then $numrej := numrej + 1$

• Garbage collect CT: (do when $ltimes$ has exactly S entries)
* $maxltime :=$ the $(H+1)$-st largest element in $ltimes$
* $ltimes :=$ empty set
* if $maxltime > ltime\text{-}est$ then
* $ltime\text{-}est := 2^j$, where $j = \min\{k : 2^k \geq maxltime\}$
* else if $(numacc > p \cdot numrej)$ and $(ltime\text{-}est > maxltime > 0)$ then
* $ltime\text{-}est := 2^j$, where $j = \min\{k : 2^k \geq maxltime\}$
* $numacc := numacc - p \cdot numrej - 1$
* $numrej := 0$
for $i = 1$ to n do
if $(CT[i] \neq nil)$ and $(CT[i] \leq clock - ltime\text{-}est)$ then
$upper := \max(upper, CT[i])$
$CT[i] := nil$

Figure 3: The limited horizon algorithm

extra accepted messages to counteract temporary surges in the number of summary-rejected messages.

The analysis of the limited horizon algorithm parallels that of the unlimited horizon algorithm. It depends on the notion of a *normal* message, a message whose lifetime is not among the H highest lifetimes in the multiset *ltimes* when garbage collection occurs. As before, Condition 1 holds by Theorem 2. Lemma 7, analogously to Lemma 3, gives an upper bound on the time after the estimate stabilizes until no more normal messages are summary-rejected. Theorem 8 states that eventually the estimate converges, and is proved similarly to Theorem 4 using Lemma 7. Lemma 9, analogously to Lemma 5, gives a constant upper bound on the number of normal messages that can be summary-rejected in any interval. (The constant is slightly different however.) It is used in the proof of Theorem 10, which states that the ratio of lost to accepted messages approaches $1/p$.

To show the desired liveness behavior of the limited horizon algorithm in (t, X, S, H)-bounded executions, we need a modified version of Lemma 3, which only refers to normal messages.

Lemma 7 *Let t_1 be any garbage-collection time in any execution of the limited horizon algorithm. Let B be the new value of ltime-est at time t_1. If t_2 is a garbage-collection time such that $t_1 + B < t_2$, and ltime-est is non-increasing in the closed interval $[t_1, t_2]$, then no normal message is summary-rejected in $[t_1 + B, t_2]$.*

Proof: The proof is essentially the same as the proof of Lemma 3. ■

The next theorem is the analog of Theorem 4.

Theorem 8 *Consider any execution of the limited horizon algorithm. Let t and X be such that the execution is (t, X, S, H)-bounded. Then there exists some real time $t' > t$ such that the value of the variable ltime-est at every time after t' is less than $2X$.*

Proof: The proof is essentially the same as the proof of Theorem 4. ■

To show that L_u/A_u converges to $\delta = 1/p$ in any (t, X, S, H)-bounded execution, we must modify Lemma 5 to refer only to normal messages. (The definitions of I_i, r_i, and a_i are the same as before.)

Lemma 9 *Consider any execution of the limited horizon algorithm that is (t, X, S, H)-bounded for some t and X. Then there exists a constant Z (depending on X) such that for all i with I_i beginning after t, the number of normal messages summary-rejected in I_i is at most Z.*

Proof: The proof is essentially the same as the proof of Lemma 5, except that Z is equal to $(\lceil \log X \rceil + 1)(2XR + S)$, since S is the maximum number of messages received in any garbage-collection period. ■

Theorem 10 *Consider any execution of the limited horizon algorithm. Let t and X be such that the execution is (t, X, S, H)-bounded. Then the limit of L_u/A_u as u tends to infinity is $1/p$.*

Proof: We show that for the fixed p, there exists t_0 such that $L_u \cdot p \leq A_u$ for all $u \geq t_0$.

Pick any time u, and let i be such that u is in interval I_i and I_i begins after time t. Let k be such that u occurs in the k-th garbage-collection period of I_i. By Lemma 9,

$$L_u \leq r_1 + \cdots + r_{i-1} + Z + k \cdot H$$

and

$$A_u \geq a_1 + \cdots + a_{i-1} + (k-1) \cdot (S - H) - Z.$$

Since $p \leq (S - H)/H$ and $a_j = p \cdot r_j + 1$ for all j,

$$L_u \cdot p \leq A_u - (i-1) + (p+1) \cdot Z + S - H.$$

Note that $i - 1$ tends to infinity, while $(p+1) \cdot Z + S - H$ is a constant. Thus there is some i_0 such that for all $i \geq i_0$, $i - 1 \geq (p+1) \cdot Z + S - H$. Thus for all times u starting with interval i_0, $L_u \cdot p \leq A_u$. ■

4 Lower bound

The proofs of Theorems 4 and 8 give us some information about how long it takes the estimate *ltime-est* to converge to less than twice the actual upper bound. In both cases, the time required for the estimate to converge depends on how long it takes to accept a certain number of messages. It might be more desirable for this amount of time to be a constant, but in this section we show that that is not possible. Specifically, we show that for any standard-form algorithm, there can be no bound, based on the previous history, between the time t after which the lifetime is at most X and the time t' after which the estimate is less than $2X$.

To state the next theorem, we need two more definitions. Let q_u be the state of the server at time u, for any u. Let M be such that in any garbage-collection period in any execution, it is possible for the server to receive at least M messages. In general, each garbage-collection period may have a different upper bound on the number of messages that can be received; M is the minimum of all these upper bounds. (For the unlimited horizon algorithm as presented, since no lower bound is specified on the time between garbage collections, technically M is 0. However, if garbage collections were evenly spaced G apart, then M would be RG. For the limited horizon algorithm, in every garbage-collection period exactly S messages are received; thus M is S.)

The theorem actually only holds for algorithms that try to achieve a ratio of lost to accepted messages that is smaller than M. Since M is a positive integer, any reasonable algorithm would achieve a ratio much smaller than this.

Theorem 11 *Choose any standard-form algorithm satisfying the three conditions for $H = 0$ and any $\delta < M$. Let B be any function mapping states of the server to integers. Then there exists for some t, X, a (t, X)-bounded execution of the algorithm in which $t' - t > B(q_t)$, where t' is the time after which the estimate is always less than $2X$.*

Proof: Suppose, for contradiction, that in any (t, X)-bounded execution for any t and X, the time t' after which the estimate is less than $2X$ is such that $t' - t \leq B(q_t)$. We construct an infinite execution that violates Condition 3 as follows.

Let e_0 be any execution that is (t_0, X)-bounded for some t_0 and X and in which no messages are received in the interval $[t_0, t_0 + B(q_{t_0})]$. Suppose execution e_{i-1}, $i > 0$, is defined so that it is (t_{i-1}, X)-bounded for some t_{i-1} and no messages arrive in $[t_{i-1}, t_{i-1} + B(q_{t_{i-1}})]$. Let t'_{i-1} be the time in that interval after which *ltime-est* is less than $2X$.

Let e_i be an execution that branches off from e_{i-1} at time t'_{i-1} in which M messages with lifetimes equal to $2X$ arrive in the next garbage-collection period and are lost. Furthermore, suppose that there is a subsequent time t_i at which a message with lifetime equal to X arrives, no messages arrive in the interval $[t_i, t_i + B(q_{t_i})]$, and e_i is (t_i, X)-bounded.

Let e be the limit of e_i as i approaches infinity. Note that e is $(t_0, 2X)$-bounded and in each interval $[t'_i, t'_{i+1}]$ one message is accepted and M messages are lost. Thus in e, L_u/A_u approaches M as u increases. Since $M > \delta$, we have a contradiction. ■

A similar theorem can be shown even for a bound that does depend on the history after time t, as long as it is independent of the number of messages received after time t.

Acknowledgment

We thank Ernst Biersack, David Feldmeier, Abel Weinrib, the conference reviewers, and the journal referees for helpful comments on earlier versions of this paper, and Liuba Shrira and Greg Bollella for enlightening conversations. The comments of Abel Weinrib inspired us to develop the limited horizon algorithm.

References

[1] Comer D: Internetworking with TCP/IP: Principles, protocols, and architecture. Prentice Hall, Englewood Cliffs 1988

[2] Herlihy MP, McKendry M: Timestamp-based orphan elimination. IEEE Transactions on Software Engineering 15(7):825–831 (1989)

[3] Jain R: Divergence of timeout algorithms for packet retransmissions. Proc 5th Annual International Phoenix Conference on Computers and Communications 1986, pp 174–179

[4] Lamport L: Time, clocks, and the ordering of events in a distributed system. Commun ACM 27(7):558–565 (1978)

[5] Lamport L, Melliar-Smith P: Synchronizing clocks in the presence of faults. J ACM 32(1):52–78 (1985)

[6] Liskov B: Practical uses of synchronized clocks. Proc 10th Annual ACM Symposium on Principles of Distributed Computing 1991, pp 1–9.

[7] Liskov B, Shrira L, Wroclawski J: Efficient at-most-once messages based on synchronized clocks. ACM Trans Computer Systems 9(2):125–142 (1991)

[8] Mills DL: Internet time synchronization: The network time protocol. IEEE Transactions on Communications 39(10):1482–1493 (1991)

[9] Steiner JG, Neuman BC, Schiller JI: Kerberos: An authentication service for open network systems. Usenix Conference Proceedings 1988, pp 191-202

[10] Zhang L: Why TCP timers don't work well. Proc ACM SIGCOMM Symposium 1986, pp 397–405