Chapter 5

Leader Election

The goal of the Leader Election (LE) algorithm is for a group of processes to select one of the processes to serve as its *leader*. LE is a classic distributed system problem that arises in many different venues. It has been used to establish a coordinator for re-organization after system failures [31], a central lock coordinator in distributed databases [43], a primary site for a distributed file system [5], k-resiliency in a reliable distributed system that can withstand k process failures [12], and the lead router to track host interest in multicast groups [13].

The goal is to converge to exactly one leader – no more, no less. Therefore, all processes must recognize the same process as the leader. They accomplish this by using a global leader selection criterion, for example by picking the process with the largest address, the lightest load, the most neighbors, the earliest wake-up time, etc. It doesn't matter what the criterion, so long as the same comparison is done by all processes and it results in a single leader. Although such an algorithm could be used to elect k multiple leaders, e.g., the processes with the k largest addresses within the group, we do not consider that problem in this thesis.

The challenge in a loosely-coupled domain is for processes to agree upon a given leader without using strict consensus. Instead of using acknowledgment-based messaging, processes send announcement-style messages as with SUP and AL. The point is for the group to converge upon a single leader after a small number of rounds of announcements, and, although LE may lead to periods of uncertainty due to transmission delay and packet loss, for leadership to stabilize eventually. In this chapter, we analyze the probabilistic consensus reached through LE and the resulting performance tradeoffs.

We explore two forms of leader election: the most basic, Leader Election using Announce-Listen (LE-A); and Leader Election using Suppression (LE-S), a refinement to the basic algorithm that inserts a phase of suppression before a leader begins making announcements. By contrasting these two methods, we unequivocally show the importance of Suppression as a scalability technique for group communication.

We identify several metrics that characterize the performance of these algorithms: Leadership

Delay, the delay until leadership is established; Leadership Re-establishment Delay, the delay until leadership is re-established after group membership changes; the Number of Messages Generated by LE; and Inconsistent State, the fraction of time processes are in disagreement about the leadership. We study the behavior of the metrics with no loss, correlated loss and uncorrelated loss. We also examine their behavior when all processes begin simultaneously, when group membership remains constant, when processes join and when they depart.

It is common practice for Leader Election algorithms to define the leader as the process with the greatest address. Although it is a widely used approach in many LE algorithms [31] [2] [25] [16] [48] [13], we reveal through analysis and simulation the critical need to consider alternate leader selection criteria for multicast-based Leader Election algorithms.

What is particularly interesting about Leader Election is that it is composed out of other techniques or slight variations of them. Consequently, we can assess its metrics in terms of the metrics for components we already have analyzed. Therefore, we compare the performance metrics for Leader Election with the bounds we have already established for the Suppression and Announce-Listen algorithms.

Before concluding this chapter, we present a summary of our findings, contrast our research with prior art, and present ideas for future work in this area.

5.1 Basic Algorithm

We explore two approaches to Leader Election. They vary in terms of how a process begins the algorithm.

If all processes were to wait to hear an announcement from the leader, then all processes might end up waiting indefinitely, with none making forward progress. Therefore, a process eventually must declare leadership, even if leadership only lasts temporarily. In both of the LE algorithms that we examine, when a process starts participating, it always assumes it is the leader, until such time as it hears an announcement from another process that challenges its leadership.

In the simplest form of the algorithm, Leader Election using Announce-Listen (LE-A), processes announce their leadership immediately, whereas with the Leader Election using Suppression (LE-S) form of the algorithm, processes delay leadership announcements until a Suppression phase completes.

In either case, while a process believes it is the leader, it announces this fact periodically to the rest of the processes. However, during times of flux, there may be multiple processes that believe they are leader. When a process that believes it is a leader receives an announcement from another process, a conflict resolution algorithm is needed. For the purpose of the examples throughout the chapter, we will use the standard method of conflict resolution that chooses a leader based on greatest address. Later in the chapter we also discuss the benefits of using alternate leader selection criteria.

Thus, if the announcement message received has a greater sender address than the process receiving the message, then the receiver gives up leadership. The receiving process defers to the new leader by refraining from sending further announcements. If a non-leader process no longer receives leadership announcements, it times out and begins the algorithm anew.

5.1.1 The Simplest Case

The simplest Leader Election algorithm is described by the pseudocode in Program 5.1.1 and the accompanying state transition diagram as shown in Figure 5.1. As with earlier state transition diagrams, we use the convention that state transitions are labelled with $\frac{A}{B}$, where A is an event that occurs, and B is the consequence of the event occurring.

```
LEADER-ANNOUNCE ($T_A$, $T_L$)
1   $leader = me.addr$
2   announce ($I\ am\ leader$)
3   set_announce_timer ($T_A$)
4   do
5       if receive_message ($msg$) then
6           if $msg.addr > leader$ then
7               $leader = msg.addr$
8               set_listen_timer ($T_L$)
9               clear_announce_timer ()
10      if announce_timer_expired () or
11          listen_timer_expired () then
12          $leader = me.addr$
13          announce ($I\ am\ leader$)
14          set_announce_timer ($T_A$)
```

Program 5.1: **Leader Election Algorithm using Announce-Listen.**

We refer to this form of the algorithm as Leader Election using Announce-Listen (LE-A). Each process begins by immediately announcing its own leadership, then setting the announce timer, T_A, to remind itself when to send the next announcement. A process then waits for one of several events to occur:

1. If an announcement is received from another process with an address greater than the current leader, the process updates the leader address and sets the listen timer, T_L, which will detect when the leader departs. If the receiving process was the old leader, it reverts to listening and clears its announce timer, as it no longer should be sending announcements.

2. If the announce timer expires, the process reaffirms its leadership, sends an announcement to the other processes, then resets the announce timer again.

If the listen timer expires, the previous leader has stopped announcing leadership, either because it was preempted by another process or because it has left the system. At that point, the leader election procedure is restarted. Any process that detects leaderlessness appoints itself leader, sends a leadership announcement and resets its announce timer, just as if it had been the leader all along.

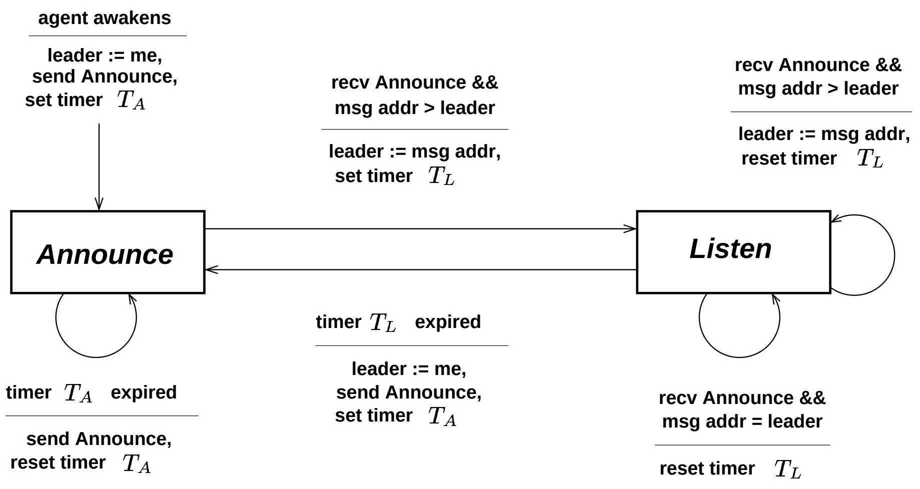
agent awakens
leader := me, send Announce, set timer T_A
recv Announce && msg addr > leader
leader := msg addr, set timer T_L
recv Announce && msg addr > leader
leader := msg addr, reset timer T_L
Announce
Listen
timer T_L expired
leader := me, send Announce, set timer T_A
timer T_A expired
send Announce, reset timer T_A
recv Announce && msg addr = leader
reset timer T_L

Figure 5.1: **Announce Leadership.**

If every process starts at the same time, each will propose itself as the leader and the algorithm will generate N messages, where N is the number of processes participating in the algorithm. In the lossless case, an agreed-upon leader will be elected after a single round of the algorithm. However, when messages can be lost, multiple rounds of conflict resolution will be necessary to reduce the number of leaders and to converge on a single leader.

5.1.2 Leader Election Refined

To avoid excessive message generation as well as message collision when all processes awaken simultaneously, each process may suppress sending its initial leadership announcement. Thus, this form of the LE algorithm combines Announce-Listen with Suppression. To differentiate it from the previous algorithm, we refer to it as Leader Election with Suppresion (LE-S). The pseudocode for the algorithm is given by Program 5.2 and the accompanying state transition diagram is shown in Figure 5.2.

Before a process issues its first message, it selects a random time t from the distribution d using a Suppression timer interval T_S, and sets the suppress timer. If an announcement *with a greater address* has not been received by time t, the process assumes leadership and begins to send announcements. A process then waits for one of several events to occur:

1. If an announcement is received from another process with a greater address than the current leader, the process updates the leader address and sets the listen timer, T_L, which will detect when the leader departs. The process begins listening for heartbeat messages from the leader.

   If the receiving process was the old leader, it reverts to listening and clears its announce timer, as it no longer should be sending announcements. The process also clears its suppress timer, in case the message was received while the process was waiting for its suppression timer to expire.

2. If the listen timer expires, the leader election process begins anew. The local process selects a random time t to sleep and resets the suppress timer.

3. If the announce timer expires or the suppress timer expires, the process reaffirms leadership, sends an announcement, then resets the announce timer. If the suppress timer expires, then we know that no other process challenged the process' leadership in the last t units of time.

```
LEADER-SUPPRESS ($T_A$, $T_L$, $T_S$ $d$)
    $leader = me.addr$
    $t$ = random ($d$, $T_S$)
    set_suppress_timer ($t$)
    do
        if receive_message ($msg$) then
            if $msg.addr > leader$ then
                $leader = msg.addr$
                set_listen_timer ($T_L$)
                clear_suppress_timer ()
                clear_announce_timer ()
        if listen_timer_expired () then
            $leader = me.addr$
            $t$ = random ($d$, $T_S$)
            set_suppress_timer ($t$)
        if suppress_timer_expired () or
            announce_timer_expired () then
            announce ($I\ am\ leader$)
            set_announce_timer ($T_A$)
```

Program 5.2: **Leader Election Algorithm using Suppression.**

The main difference from Suppression as discussed in Chapters 2 and 3 is that a process is not suppressed by the arrival of a message from merely another process; the message must have originated from a process with a greater address.

5.1.3 A Note about Timers

The Leader Election algorithm relies on several timer values: an Announce timer, T_A, represents the periodicity of leadership announcements; a Listen timer, T_L, represents the interval after which

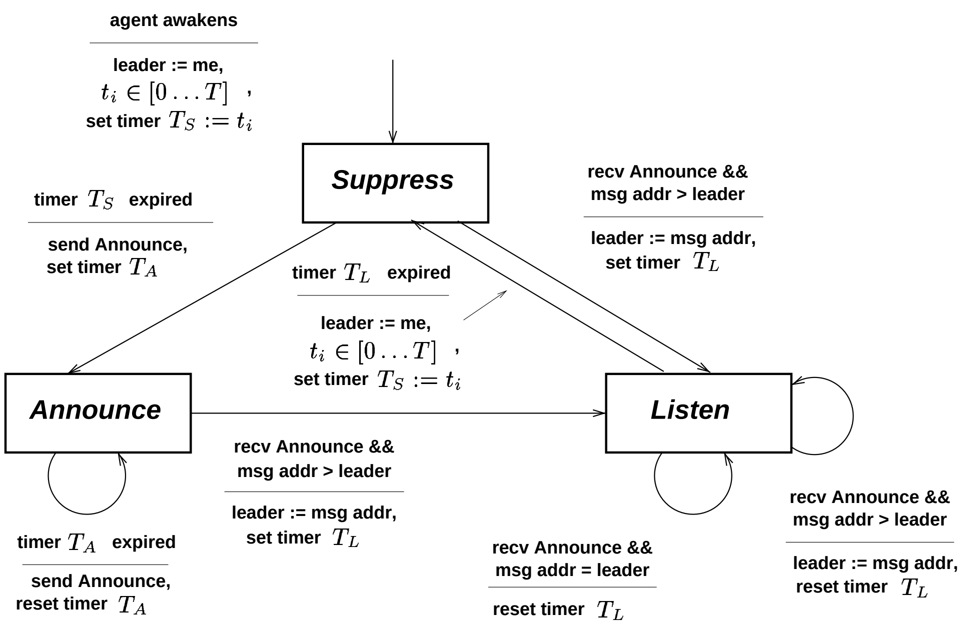
agent awakens
leader := me, $t_i \in [0 \ldots T]$, set timer $T_S := t_i$
Suppress
timer T_S expired
send Announce, set timer T_A
recv Announce && msg addr > leader
leader := msg addr, set timer T_L
timer T_L expired
leader := me, $t_i \in [0 \ldots T]$, set timer $T_S := t_i$
Announce
Listen
recv Announce && msg addr > leader
leader := msg addr, set timer T_L
timer T_A expired
send Announce, reset timer T_A
recv Announce && msg addr = leader
reset timer T_L
recv Announce && msg addr > leader
leader := msg addr, reset timer T_L

Figure 5.2: **Suppress Leadership.**

a process decides it has missed an announcement it was expecting[1]; a Suppress timer, T_S, is the upper bound of the Suppression interval from which a process selects a time to awaken.

Note that T_L is typically a small integer multiple of T_A, and T_S is meant to be large enough for most processes to hear a leadership announcement before sending their own, yet small enough not to delay consensus on the true leader.

5.2 Scalability

The way in which the Leader Election algorithm provides scalability is to appoint (ideally) a single leader process that acts on behalf of the other processes. As the representative of the group, the leader issues messages, so that the other members do not have to. In addition, LE offers all of the benefits already enumerated for Announce-Listen, as well as the benefits of Suppression in the case of LE-S: it reduces the number of overall messages generated, decreases the likelihood of implosion, adds asynchrony to group transmissions, removes the tight feedback loop required for processing ACKs and NACKs, and eliminates the delay waiting for their retransmission.

5.3 Metrics

To evaluate the performance of Leader Election, we explore several possible metrics. The metrics are reminiscent of the metrics derived to evaluate SUP and AL, but they are recast in terms of

[1]As in the AL algorithm, we could gradually age leaders, but for simplicity we are not doing that here.

leadership establishment. Although not a complete list, the ones below are representative of the kinds of performance issues that are important for efficient operation of LE.

- **Leadership delay.** Leadership delay is the amount of time it takes until an agreed-upon leader is established. Leadership is established when the announcement message sent by the process with the largest address has been received by all other group members. This metric determines the average time taken by the LE process to complete or to stabilize.

- **Leadership re-established delay.** Leadership must be re-established after a perturbation in the system, such as when the leader leaves, a new leader arrives, or when messages lost in the network cause a temporary outage of communication between the leader and other processes. To be precise, leadership is re-established when all processes agree upon the new leader. This metric determines the average time taken by the LE process to re-stabilize after a perturbation in membership.

- **Number of messages generated.** When a process is a leader, it periodically announces its leadership. The number of messages generated in a given interval (of size T_A) equals the number of processes announcing leadership during that time.

- **Number of simultaneous leaders.** We can estimate the number of leaders in a given interval by determining the number of announcement messages generated in that interval. Therefore, we focus on the number of messages generated and use that as an estimate.

- **Inconsistent state.** A process could have inconsistent state in three cases: (a) when the process that has been elected as the leader has left the system, but leadership has not been re-estabilished yet, (b) when a new leader process arrives into the system, but all processes have not yet reached agreement on this fact, or (c) a process falsely believes the leader has departed (because its listen timer expires) but the leader has not actually left.

In the upcoming sections, we discuss the performance of these metrics when there is: no loss, correlated loss and uncorrelated loss. We consider each metric under the condition that all processes begin simultaneously, as it presents the most stress on the algorithm's attempt to moderate the numbers of announcing processes. For the re-establishment delay metric, we consider additional scenarios: all processes are in a *steady state* (i.e., they remain in agreement about the leader), processes join the algorithm, and processes depart.

Our model of the network is identical to that used in previous chapters, with parameters for group size, N, loss probability, l, and fixed transmission delay between processes, Δ. Unless indicated, the analysis assumes all processes use the LE-S algorithm, since LE-A can be derived from LE-S by setting $T_S = 0$.

It is also of particular interest to us that LE-S is a combination of Suppression and Announce-Listen. As such, we posit that its performance is bounded in the best case by the performance of SUP and AL. We show these comparisons in Section 5.4.

Because LE is not strictly the composition of SUP and AL, but rather a composition of variants of these algorithms, at times it behaves in a manner that is quite different from them. For example, in LE there is *always* a leader, as long as there are processes participating in the algorithm. As such, it is impossible to find the corollary of AL errors for false positives and false negatives in this context. A false positive would occur when a process has the false belief that there is a leader, when there is none. A false negative would occur when a process has the false belief that there is no leader, when there is actually a leader present. Since leadership is immediately established when the LE algorithm begins, and always reverts back to each local process when the leader departs, there is always a leader in the LE algorithm, and hence these conditions (of false positives and false negatives) do not arise in LE. However, when a process falsely believes the leader has departed (because it has not received a leader announcement message within the timeout period), and thus reverts back to itself as the leader, we consider this condition under the inconsistent state metric.

5.3.1 Leadership Delay

What is the expected delay until leadership is established? In other words, at what point in time does the number of leaders converge to exactly one? The assumption is that all processes begin simultaneously at time $t = 0$. We examine the case where processes arrive at later times ($t > 0$) in Section 5.3.2 when we explore re-establishment delay. We also assume that $N > 1$ otherwise there is no leadership delay. If one and only one process arrives into the system, it is immediately the leader as it requires no agreement to be reached with any other processes.

5.3.1.1 No Loss

In the case of the simple LE algorithm (LE-A), the system begins with N leaders when all N processes begin simultaneously. After time Δ has elapsed, only one leader remains.

$$E[\textit{leadership delay LE-A}]_{l=0} = \Delta$$

In the case of Leader Election with Suppression (LE-S), leadership will be established at time $t_s + \Delta$, where t_s is the Suppression wake-up time selected by the process with the largest address. On average, this time will be $E[t_s] + \Delta$, where $E[t_s] = \int_0^{T_S} tp(t)\,dt$.

$$E[\textit{leadership delay LE-S}]_{l=0} = E[t_s] + \Delta$$

5.3.1.2 Correlated Loss

Message loss in the network will cause larger leadership delay because it may take multiple announcements before an announcement from the leader reaches all processes. With correlated loss, if a message is lost, it is lost by all receivers. Conversely, if a message is received by a receiver, it is received by all receivers. The question is, how many announcements will it take before the leader's announcement is successfully received? Let i be the announcement number, where $i = 0$ corresponds to the first announcement sent by the leader. Then, the expected time for leadership to be established is given by:

$$E[\mathit{leadership\ delay\ LE\text{-}A}]_{corr} = \sum_{i=0}^{\infty} Pr[\mathit{announcement}\ i\ \mathit{establishes\ leadership}]_{corr} \times (\Delta + T_A \times i)$$

The likelihood of the first ($i = 0$) announcement from the leader being received by all processes is $(1 - l)$; in general, the likelihood of the i^{th} announcement establishing leadership is the probability that the previous announcements were lost but the i^{th} announcement was received, which is $l^i(1-l)$. Therefore, the expected leadership time is given by:

$$\begin{aligned} E[\mathit{leadership\ delay\ LE\text{-}A}]_{corr} &= \sum_{i=0}^{\infty} l^i(1-l)(\Delta + T_A \times i) \\ &= \sum_{i=0}^{\infty} l^i(1-l) \times \Delta + \sum_{i=0}^{\infty} l^i(1-l) \times T_A \times i \\ &= \Delta + \sum_{i=0}^{\infty} l^i(1-l) \times T_A \times i \\ &= \Delta + \frac{T_A l}{(1-l)} \end{aligned}$$

When we have suppression, the only difference in this metric is that the leader process starts announcing at a time selected by the suppression algorithm. Therefore,

$$\begin{aligned} E[\mathit{leadership\ delay\ LE\text{-}S}]_{corr} &= E[t_s] + E[\mathit{leadership\ delay\ LE\text{-}A}]_{corr} \\ &= E[t_s] + \Delta + \frac{T_A l}{(1-l)} \end{aligned}$$

5.3.1.3 Uncorrelated Loss

With uncorrelated loss, different processes may lose different announcements from the leader. Thus, the difference between the delay calculation for correlated and uncorrelated loss is in the last term of the leadership delay formula. This term establishes the expected number of rounds it takes for the leader's announcement message to reach all processes. A detailed analysis of that difference can be found in Appendix B. Essentially, we establish that the probability for all N processes to have received a message in less than or equal to i attempts is $(1 - l^i)^N$. As a result, we can derive the probability that all N processes have received the message in exactly i attempts is $(1 - l^i)^N - (1 - l^{(i-1)})^N$, the difference between the probability that all processes have received the message in less than or equal to i tosses and the probability that all processes have received the message in less than or equal to $i - 1$ tosses. We incorporate the results below.

$$\begin{aligned}
E[\textit{leadership delay LE-S}]_{uc} &= E[t_s] + \Delta + \\
&\quad \sum_{i=0}^{\infty} Pr[\textit{announcement } i \textit{ establishes leadership}]_{uc} \times T_A \times i \\
&= E[t_s] + \Delta + \sum_{i=0}^{\infty} \left((1 - l^i)^N - (1 - l^{(i-1)})^N\right) \times T_A \times i \\
&= E[t_s] + \Delta + T_A \times \sum_{i=1}^{N} (-1)^{(i+1)} \binom{N}{i} \frac{1}{(1 - l^i)}
\end{aligned}$$

5.3.2 Leadership Re-establishment Delay

The leadership re-establishment process is begun after either the departure of the leader or the arrival of a new process into the system. Departures are detected by the expiration of the listen timer, T_L. However, the listener cannot discriminate between when the leader actually departs or when the leader is falsely thought to have departed but has not, i.e., if the leader's announcement messages are delayed or lost in the system. We calculate the re-establishment delay under both circumstances, referring to the former case as *leader departure* and the latter as *false departure.* We refer to arriving processes as *late joiners* and assume they occur during the steady state, after there is already an established leader.

The calculations below also make the assumption that $N \geq 1$ after the leader departs and $N \geq 1$ before a process arrives. We assert that if the last process departs the system, then there is no re-establishment delay. Similarly, when the first and only process arrives into the system, there is no re-establishment delay. It becomes the leader immediately. Note that the departure of a non-leader process has no effect on the consensus of leadership among the remaining processes.

5.3.2.1 No Loss

Leader Departure. Consider the scenario depicted in Figure 5.3. The leader sends its k^{th} and last announcement at time kT_A. All listeners receive this heartbeat message from the leader at time $kT_A + \Delta$. The leader subsequently departs at some point, $kT_A + t_d$, within the announcement interval $[kT_A,\ (k+1)T_A]$. We calculate $E[t_d] = \int_0^{T_A} t \cdot D(t)dt$ from the departure function, $D(t)$ and

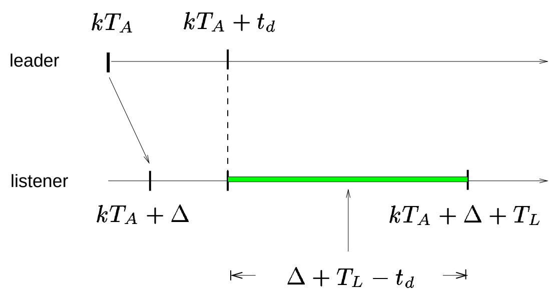

Figure 5.3: **Time to Notice the Leader Left (No Loss).**

assume $D(t)$ is Poisson, meaning in each interval it behaves similarly.[2]

The listeners time out after T_L and subsequently perform the Suppression phase of the algorithm, which amounts to the normal leadership establishment delay. The total time to notice that the leader left:

$$\begin{aligned} E[\textit{time to notice leader left}]_{l=0} &= E[(kT_A + \Delta + T_L) - (kT_A + t_d)] \\ &= \Delta + T_L - E[t_d] \end{aligned}$$

The total re-establishment delay is the time to notice the leader left plus the amount of time that the listener takes to hear from the replacement leader, $t_s + \Delta$, which is the time the new leader picks for its suppression timer plus the delay before which the listeners receive its leadership announcement. If the average time a leader picks for its suppression timer is $E[t_s]$, and the expected departure time is $E[t_d]$, the expression becomes:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{l=0} &= E[\textit{time to notice leader left}]_{l=0} + \\ &\quad E[\textit{leadership delay LE-S}]_{l=0} \\ &= (\Delta + T_L - E[t_d]) + (E[t_s] + \Delta) \\ &= T_L + 2\Delta + E[t_s] - E[t_d] \end{aligned}$$

False Departure. For leaderlessness to have been detected, the listen timer must have expired. The expiration of this timer occurs T_L after the last announcement was received from the leader. In the no loss case, this only occurs if the timer expired before the next announcement had a chance to arrive. The implication is that $T_L < T_A$, since we are assuming fixed transmission delay Δ. At the point when the timer expires, the local process reverts to itself as leader and begins the election

[2]Otherwise $E[t_d]$ would need to be averaged over all intervals: $E[t_d] = \lim_N^\infty (1/N)(\int_{T_A}^{0} D(T)dt + \int_{2T_A}^{T_A}(t - T_A) \cdot D(T)dt + \ldots + \int_{NT_A}^{(N-1)T_A}(t - (N-1)T_A) \cdot D(T)dt)$.

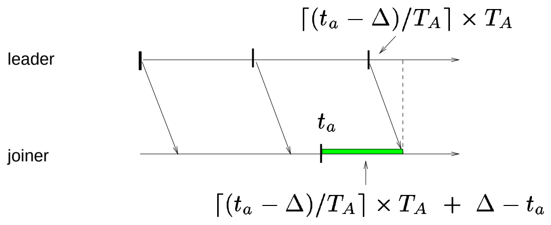
$\lceil (t_a - \Delta)/T_A \rceil \times T_A$
leader
t_a
joiner
$\lceil (t_a - \Delta)/T_A \rceil \times T_A \ + \ \Delta - t_a$

Figure 5.4: **Leadership Delay with a Late Joiner:** *joiner $\neq$ leader*

process anew. Therefore, the re-establishment delay becomes the portion of the announcement interval that remains until the next announcement arrives,

$$E[\textit{re-establishment delay LE-S}]_{l=0,false} = T_A - T_L$$

Late Joiners. The system reacts differently to a late joiner depending on whether its address is smaller or larger than the current leader. If the late joiner has a larger address than the current leader and it arrives into the system at time t_a, then it will become the new leader at time $t_a + \Delta$. We calculate $E[t_a] = \int_0^{T_A} t \cdot A(t)dt$ from the arrival function $A(t)$ and assume A(t) is Poisson. We subtract the arrival time from the time when joiners' leadership announcement reaches the other processes,

$$\begin{aligned} E[\textit{re-establishment delay LE-A}]_{l=0,joiner>leader} &= E[t_a] + \Delta - E[t_a] \\ &= \Delta \end{aligned}$$

If the late joiner has a smaller address than the current leader, the late joiner must receive a message from the current leader before leadership is fully re-established (all group members agree on the leader), which happens when the next announcement made by the current leader reaches the late joiner. The previous announcement must have occurred before time $t_a - \Delta$; otherwise the late joiner would have been suppressed immediately on arrival. If the number of announcements that have occurred at a given point in time, t, is t/T_A, then the time until the late joiner receives the next announcement is $\lceil (t_a - \Delta)/T_A \rceil \times T_A + \Delta$. Therefore, the leadership delay is as displayed in Figure 5.4:

$$E[\textit{re-establishment delay LE-A}]_{l=0,joiner\neq leader} = E[\lceil (t_a - \Delta)/T_A \rceil] \times T_A + \Delta - E[t_a]$$

In the case of Leader Election with Suppression (LE-S), when a late joiner with a larger address arrives at time t_a, then it will become the new leader and begin announcing leadership at time $t_a + t_s$. The time at which the announcements reach the other processes is $t_a + t_s + \Delta$. The leadership delay is the same as if all processes began simultaneously:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{l=0,joiner>leader} &= (E[t_a]+E[t_s]+\Delta)-E[t_a] \\ &= E[t_s]+\Delta \\ &= E[\textit{leadership delay LE-S}]_{l=0} \end{aligned}$$

When a late joiner with a lower address arrives at time t_a, then the delay is equivalent to the same scenario in the LE-A case:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{l=0,joiner\neq leader} &= E[\lceil(t_a-\Delta)/T_A\rceil]\times T_A+\Delta-E[t_a] \\ &= E[\textit{re-establishment delay LE-A}]_{l=0,joiner\neq leader} \end{aligned}$$

5.3.2.2 Correlated Loss

Leader Departure. When there is loss in the system, the leader could depart **after** the listener timer has begun expiration already due to lost messages. When we examine the average number of consecutive lost messages, we find that

$$\begin{aligned} E[\textit{\# consecutive lost}]_{corr} &= E[\textit{announcement i establishes leadership}]_{corr} \\ &= \frac{l}{(1-l)} \end{aligned}$$

The time to notice the leader left is basically shifted by the term ($T_A \times E[\textit{\# consecutive lost}]$), as is the leadership delay we already derived in Section 5.3.1.2. Therefore,

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{corr} &= E[\textit{time to notice leader left}]_{corr} + \\ &\quad E[\textit{leadership delay LE-S}]_{corr} \\ &= (\Delta+T_L-E[t_d])-(T_A\times E[\textit{\# consecutive lost}]_{corr})+ \\ &\quad (E[t_s]+\Delta+\frac{T_A l}{1-l}) \\ &= T_L+2\Delta-E[t_d]+E[t_s] \\ &= E[\textit{re-establishment delay LE-S}]_{l=0} \end{aligned}$$

Note that this is the same result as the lossless case! Basically, the gains made in the time to notice the leader left are negated by the increases in leadership delay. They cancel each other out.

False Departure. For leaderlessness to be detected, the listen timer at a receiver process must have expired T_L beyond the last successfully received announcement from the leader. In the lossy case, this arises when the listen timer is not long enough to withstand multiple message drops, i.e., $T_L \leq T_A \times E[\# \textit{ consecutive lost}]$. The detection of leaderlessness happens T_L units into the interval $T_A \times E[\# \textit{ consecutive lost}]$, and is corrected one announcement beyond that interval, so the re-establishment delay is:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{corr,false} &= T_A \times E[\# \textit{ consecutive lost}]_{corr} - T_L + T_A \\ &= T_A \times \left(\frac{l}{1-l} + 1\right) - T_L \\ &= T_A \times (\frac{1}{1-l}) - T_L \end{aligned}$$

Late Joiners. When the late joiner with a larger address than the current leader arrives at time t_a, then it will become the new leader and begin announcing at time $t_a + t_s$. However, for all processes to agree on the leader, the joiner's announcement must reach all processes. With loss, we know on average that the number of announcements it will take for that to occur is $E[\# \textit{ consecutive lost}]_{corr}$, and an additional Δ for the last announcement to reach all processes. Therefore, we use an adjustment term $(T_A \times E[\# \textit{ consecutive lost}])$ once again:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{corr,joiner>leader} &= E[t_a] + E[t_s] + \\ &\quad (T_A \times E[\# \textit{ consecutive lost}]_{corr}) + \Delta - E[t_a] \\ &= E[t_s] + \frac{T_A l}{1-l} + \Delta \\ &= E[\textit{leadership delay LE-S}]_{corr} \end{aligned}$$

When a late joiner with a lower address arrives at time t_a, then leadership is re-established once the late joiner receives the leader's announcement. In the correlated case, this may take $E[\# \textit{ consecutive lost}]_{corr}$ announcements, therefore:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{corr,joiner \neq leader} &= E[\lceil (t_a - \Delta)/T_A \rceil] \times T_A \\ &\quad + (E[\# \textit{ consecutive lost}]_{corr} \times T_A) + \Delta - E[t_a] \\ &= (E[\lceil (t_a - \Delta)/T_A \rceil] + \frac{l}{1-l}) \times T_A + \Delta - E[t_a] \end{aligned}$$

5.3.2.3 Uncorrelated Loss

The only difference between the uncorrelated and correlated loss cases is the term that indicates the number of rounds of announcements required to establish an agreed-upon leader.

Leader Departure. As can be seen from the result above, there is no loss term for this metric. Thus, all three scenarios (lossless, correlated loss and uncorrelated loss) have identical results:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{uc} &= T_L + 2\Delta - E[t_d] + E[t_s] \\ &= E[\textit{re-establishment delay LE-S}]_{l=0} \\ &= E[\textit{re-establishment delay LE-S}]_{corr} \end{aligned}$$

False Departure.

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{uc,false} &= T_A \times E[\#\ \textit{consecutive lost}]_{uc} - T_L + T_A \\ &= T_A \times \left(1 + \sum_{i=1}^{N} (-1)^{(i+1)} \binom{N}{i} \frac{1}{(1-l^i)}\right) - T_L \end{aligned}$$

Late Joiners. Adjusting the term that indicates the number of rounds of announcements required to establish an agreed-upon leader, we find the corresponding metrics for uncorrelated loss:

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{uc,joiner>leader} &= E[t_a] + E[t_s] + \\ &\quad (T_A \times E[\#\ \textit{consecutive lost}]_{uc}) + \Delta - E[t_a] \\ &= E[t_s] + \Delta + T_A \times \sum_{i=1}^{N} (-1)^{(i+1)} \binom{N}{i} \frac{1}{(1-l^i)} \\ &= E[\textit{leadership delay LE-S}]_{uc} \end{aligned}$$

$$\begin{aligned} E[\textit{re-establishment delay LE-S}]_{uc,joiner\neq leader} &= E[\lceil (t_a - \Delta)/T_A \rceil] \times T_A + \\ &\quad (E[\#\ \textit{consecutive lost}]_{uc} \times T_A) + \Delta - E[t_a] \\ &= \left(E[\lceil (t_a - \Delta)/T_A \rceil] + \sum_{i=1}^{N} (-1)^{(i+1)} \binom{N}{i} \frac{1}{(1-l^i)}\right) \\ &\quad \times T_A + \Delta - E[t_a] \end{aligned}$$

5.3.3 Number of Messages Generated

5.3.3.1 No Loss

Consider a collection of N processes, and a particular process q_k with the $(k+1)^{st}$ largest address. As shown in Figure 5.5, vector $\vec{Q}_{max}$ is an ordering of process addresses from largest to smallest. Thus, there are k processes with addresses larger than q_k. Given process q_k picks suppression time t, it will announce leadership by sending a message only if it does not receive a message from a process with a larger address. In other words, it becomes leader if all the k processes with larger addresses pick suppression times larger than $t - \Delta$. The likelihood of this event is $(1 - P(t - \Delta))^k$, where

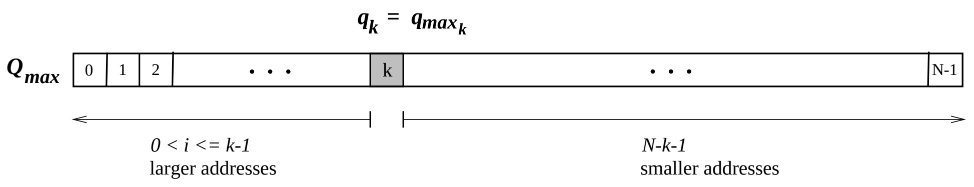

Figure 5.5: **Vector $\vec{Q}$ of Process Addresses Ordered from Largest to Smallest.**

$p(t)$ is the probability density function for the suppression timer and $P(t)$ the associated cumulative distribution function (as first appeared in Chapter 2). Therefore, the probability that process q_k sends a message is:

$$\begin{aligned} Pr[q_k \textit{ sends a message LE-S}]_{l=0} &= \int_0^{T_S} p(t)(1-P(t-\Delta))^k \, dt \\ &= P(\Delta) + \int_\Delta^{T_S} p(t)(1-P(t-\Delta))^k \, dt \end{aligned}$$

Therefore, the expected number of messages generated during the Suppression interval is given by:

$$\begin{aligned} E[\textit{number messages LE-S}]_{l=0} &= \sum_{k=0}^{N-1} Pr[q_k \textit{ sends a message LE-S}]_{l=0} \\ &= NP(\Delta) + \sum_{k=0}^{N-1} \int_\Delta^{T_S} p(t)(1-P(t-\Delta))^k \, dt \\ &= NP(\Delta) + \int_\Delta^{T_S} p(t) \sum_{k=0}^{N-1} (1-P(t-\Delta))^k \, dt \\ &= NP(\Delta) + \int_\Delta^{T_S} p(t) \frac{1-(1-P(t-\Delta))^N}{P(t-\Delta)} \, dt \end{aligned}$$

5.3.3.2 Correlated Loss

When there is loss present in the system, we need to consider the number of rounds until the leader announcement reaches the other processes, i.e., the number of leaders reduces to one. This means there will be some number of false leaders in each round that contribute to the total count. Below we calculate the number of messages generated in the initial Suppression interval.

Let t be the suppression time selected by q_k. In the correlated loss case, q_k will send a message if all the k processes with larger addresses pick suppression times larger than $t-\Delta$, or if their messages are lost. Note that when a message is lost by q_k, it is also lost by all other processes with addresses larger than it thereby eliminating the possibility that a message sent by q_i $(i < k)$ suppresses q_j where $i < j < k$. The likelihood of this event is $(1 - P(t-\Delta) + lP(t-\Delta))^k$. Therefore, the probability that process q_k sends a message is:

$$\begin{aligned}
Pr[q_k \textit{ sends a message in } T_S \textit{ LE-S}]_{corr} &= \int_0^{T_S} p(t)(1-(1-l)P(t-\Delta))^k\, dt \\
&= P(\Delta) + \int_{\Delta}^{T_S} p(t)(1-(1-l)P(t-\Delta))^k\, dt
\end{aligned}$$

Therefore, the expected number of messages generated during the Suppression interval is given by:

$$\begin{aligned}
E[\textit{number messages in } T_S \textit{ LE-S}]_{corr} &= \sum_{k=0}^{N-1} Pr[q_k \textit{ sends a message LE-S}]_{corr} \\
&= NP(\Delta) + \sum_{k=0}^{N-1} \int_{\Delta}^{T_S} p(t)(1-(1-l)P(t-\Delta))^k\, dt \\
&= NP(\Delta) + \int_{\Delta}^{T_S} p(t) \sum_{k=0}^{N-1} (1-(1-l)P(t-\Delta))^k\, dt \\
&= \frac{NP(\Delta)}{1-l} + \int_{\Delta}^{T_S} p(t) \frac{1-(1-(1-l)P(t-\Delta))^N}{P(t-\Delta)}\, dt
\end{aligned}$$

In later rounds, there may still exist multiple leaders, although we expect fewer than in the initial round (unless of course $l = 1$), and even fewer with each subsequent round. The processes behave similarly to the initial round in that they maintain their relative distances from each other in terms of when they issue announcements. Each process that did not receive a message from a process with a larger address continues to issue announcements (because it still believes it is the leader), until such time that it receives a message from process q_{N-1}.

The question becomes how many rounds occur before the algorithm stabilizes and how many messages are generated per round? We explore this more completely in the uncorrelated loss case below, as it is the case that generates the worst case performance, as we found with the Suppression algorithm (Section 3.3).

5.3.3.3 Uncorrelated Loss

To derive the number of messages when there is uncorrelated loss present, we use a similar approach to that used in the Suppression with loss chapter (See Chapter 3). Because of the dependencies between rounds of announcements, the problem is more tractable if we cast it as a recurrence relation. Because of the complexities that arise due to the interaction of message loss and transmission delay, we focus on a solution for the case when there is no transmission delay, $\Delta = 0$.

Let $Q(i, n)$ be the probability that i messages are sent by n processes $q_0 \ldots q_{n-1}$, given a particular vector of times selected $\vec{T} = (t_0, t_1, \ldots t_{n-1})$. While $\vec{Q}$ is fixed, $\vec{T}$ is random.

$Q(i, n)$ can be broken down into two disjoint parts: (a) Process $n-1$ sends one message, and processes $0 \leq j < n-1$ send $(i-1)$ messages; (b) Process $n-1$ does not sent a message, and processes $0 \leq j < n-1$ send i messages. In case (a), process q_{n-1} sends a message if for each process q_k, $k < n-1$, its message is either lost or is sent at a time after the time chosen by q_{n-1}, which occurs with probability $(l + (1-l)(1-P(t_{n-1})))^{i-1}$. In case (b), process $n-1$ does not send a message if it received one of the i messages. In other words the process did not lose all of the messages sent, each of which were either lost or not lost but arrived too late, which occurs with probability $1 - (l + (1-l)(1-P(t_{n-1})))^{i}$.

Therefore, we write:

$$\begin{aligned} Q(i,n) &= Pr[i \textit{ messages sent by } q_0, ..q_{n-1} \textit{ given } t_{n-1}] \\ &= \int_0^{T_S} Q(i-1,n-1) \times (l + (1-l)(1-P(t_{n-1})))^{i-1} + \\ &\quad Q(i,n-1) \times \left(1 - (l + (1-l)(1-P(t_{n-1})))^{i}\right) p(t)\, dt \end{aligned}$$

Now let us examine $Q(i,n,r)$, the probability that there are i messages sent by n processes in round r, where $r = 1$ is the first round. A process *survives* round r if it sent a message in that round and does not receive a message with a larger id. If a process did not send a message in round r, then it will not send a message in round $r+1$, as it has already been suppressed by a process with a larger id. If a process sends a message in round r, but receives a message with a larger id in that round, then it does not send a message in round $r+1$.

Let $S(n,r)$ be the probability that process q_{n-1} reaches round r.

$$S(n,r) \quad = \quad S(n,r-1) \times Pr[q_{n-1} \textit{survives round } r-1]$$

For process q_{n-1} to survive round r, it must lose all messages sent by processes with larger addresses, $q_0, q_1, \ldots q_{n-2}$. However, the probability of those losses are dependent on the number of messages actually sent in that round.

$$\begin{aligned} Pr[q_{n-1} \textit{survives round } r-1] &= Pr[q_0 \ldots q_{n-2} \textit{ send } 1 \textit{ message and } q_{n-1} \textit{ loses } 1] + \\ &\quad Pr[q_0 \ldots q_{n-2} \textit{ send } 2 \textit{ messages and } q_{n-1} \textit{ loses } 2] + \\ &\quad \ldots \\ &\quad Pr[q_0 \ldots q_{n-2} \textit{ send } n-1 \textit{ messages and } q_{n-1} \textit{ loses } n-1] \\ &= Q(1,n-1,r-1) \times l + Q(2,n-1,r-1) \times l^2 + \ldots + \\ &\quad Q(n-1,n-1,r-1) \times l^{n-1} \\ &= \sum_{i=1}^{n-1} Q(i,n-1,r-1) \times l^i \end{aligned}$$

Note that it does not matter if a message sent by a process with a larger address is sent earlier or later than q_{n-1}'s scheduled announcement; either way, q_{n-1} must lose it to survive the round.

Substituting back into the earlier equation,

$$S(n,r) = S(n,r-1) \times \sum_{i=1}^{n-1} Q(i,n-1,r-1) \times l^i$$

Having derived $S(n,r)$, we now extend our earlier result for $Q(i,n)$ to consider rounds. The probability of i messages being sent by n processes in round r:

$$\begin{aligned} Q(i,n,r) &= Pr[i \textit{ messages sent by } q_0,..q_{n-1} \textit{ in round } r \textit{ given } t_{n-1}] \\ &= S(n,r-1) \times \\ &\quad \Big(\int_0^{T_S} Q(i-1,n-1,r-1) \times (l+(1-l)(1-P(t_{n-1})))^{i-q}) + \\ &\quad Q(i,n-1,r-1) \times \left(1-(l+(1-l)(1-P(t_{n-1})))^i\right) p(t)\, dt \Big) \end{aligned}$$

We observe that each round is dependent on the previous round's results. When we recursively substitute $Q(i,n,r)$ back into $S(n,r)$, we find that $S(n,r)$ can be expressed entirely in terms of $Q(i,n,r)$:

$$S(n,r) = \prod_{k=1}^{r-1} \sum_{i=1}^{n-1} Q(i,n-1,k) \times l^i$$

Thus, $Q(i,n,r)$ itself can be expressed recursively:

$$\begin{aligned} Q(i,n,r) &= \prod_{k=1}^{r-2} \sum_{i=1}^{n-1} Q(i,n-1,k) \times l^i \times \\ &\quad \Big(\int_0^{T_S} Q(i-1,n-1,r-1) \times (l+(1-l)(1-P(t_{n-1})))^{i-q}) + \\ &\quad Q(i,n-1,r-1) \times \left(1-(l+(1-l)(1-P(t_{n-1})))^i\right) p(t)\, dt \Big) \end{aligned}$$

Since the leader process q_0 always sends a message, $Q(0,i,r) = 0$. We can also compute $Q(i,i,r)$, because this is the likelihood that each processes survives round r, i.e., each processes loses all messages sent by all other processes, in every round up to and including the current round. Therefore,

$$\begin{aligned} Q(0,i,r) &= 0 \\ Q(i,i,r) &= (l^1 \cdot l^2 \cdot l^3 \cdots l^{i-1})^r \\ &= (l^{(i-1)i/2})^r \\ &= l^{r(i-1)i/2} \end{aligned}$$

Thus, the expected number of messages generated during leadership election in the uncorrelated case when $\Delta = 0$ is given by:

$$\begin{aligned}
&E[\textit{number messages generated LE-S}]_{uc,\Delta=0} \\
&\quad = \sum_{r=1}^{\infty} \sum_{1 \le i < N} i \times r \times Pr[i \textit{ messages sent by } q_0, ..q_{n-1} \textit{ in round } r \textit{ given } t_{n-1}] \\
&\quad = \sum_{r=1}^{\infty} \sum_{1 \le i < N} i \times r \times Q(i, n, r)
\end{aligned}$$

5.3.4 Inconsistent State

There are three scenarios that lead to inconsistent state as discussed in the previous section on re-establishment delay (Section 5.3.2): leader departure, false departure, and late joiners.

- **Leader Departure.** When the process that has been elected as the leader leaves the system, leadership must be re-established. The time between the old leader departing and the new leader's announcement arriving at the other processes is considered time spent in an inconsistent state.
- **False Departure.** When a leader process is *thought* to depart because the listen timer T_L expires, the agreement process becomes unnecessarily perturbed. At that point, the local process that has detected leaderlessness takes back leadership and begins the LE algorithm anew. The time between when leadership reverts to the local process and subsequently transfers back to the true leader is considered time spent in an inconsistent state.
- **Late Joiners.** When a new process arrives into the system, it must become synchronized with the rest of the processes about which process is the leader. The time between the arrival of the new process and the time before which all processes receive the (possibly new) leader's announcement is considered time spent in an inconsistent state.

Any time the processes spend in disagreement about the leadership is considered inconsistent state. The $E[\textit{inconsisent state}]$ in the system is simply defined as the fraction of the time that the system is spent in an ambiguous state, trying to re-establish agreement on leadership. We sum over all such occurrences in the system, and then divide by the how long the group has been established,

$$\begin{aligned}
E[\textit{inconsistent state}] &= \frac{\sum E[\textit{re-establishment delay}]}{\textit{all of time}} \\
&= \Big(\sum E[\textit{re-establishment delay}]_{\textit{leader departure}} + \\
&\quad \sum E[\textit{re-establishment delay}]_{\textit{false departure}} + \\
&\quad \sum E[\textit{re-establishment delay}]_{\textit{late joiners}}\Big) / \textit{all of time}
\end{aligned}$$

We examine the problem in more detail by breaking down the problem into its components. If the interdeparture time from the system is T_d, then:

$$E[\textit{inconsistent state}]_{leader\ departure} = \frac{\sum E[\textit{re-establishment delay}]_{leader\ departure}}{T_d}$$

False departures only occur in our model (a) in the no loss case, when $T_L < T_A$, and (b) in the lossy case, on average when $T_L < T_A \times E[\#\ \textit{consecutive lost}]$. Therefore, the fraction of the time the system is inconsistent is,

$$\begin{aligned} E[\textit{inconsistent state}]_{l=0,false\ departure} &= 1 - T_L/T_A \\ E[\textit{inconsistent state}]_{lossy,false\ departure} &= 1 - \frac{T_L}{T_A \times (E[\#\ \textit{consecutive lost}] + 1)} \end{aligned}$$

If the interarrival time for late joiners into the system is T_a, then:

$$E[\textit{inconsistent state}]_{late\ joiners} = \frac{\sum E[\textit{re-establishment delay}]_{late\ joiners}}{T_a}$$

5.4 Analysis and Simulation

In the previous section, we derived a series of performance metrics to evaluate the Leader Election algorithm. In this section, we primarily focus on the metrics for leadership delay and the number of messages generated. We validate the analysis with simulation, comparing the behavior of LE with that of SUP and AL. We examine alternate leader selection criteria to the standard method of using the largest process identifier. We present an overview of general trends of the algorithm, contrasting the behavior between the different versions of the LE algorithm under different operating conditions.

In our simulations, the group size N remained unchanged during the algorithm, the timer intervals T_S, T_A and T_L were fixed, and the transmission delay Δ was uniform. Each simulation was run 1000 times when $N \leq 100$, and 100 times when $100 < N \leq 500$. In the Suppression phase, each node chose a delay time, t_i, based on the Unix `srandom()` function that had been seeded with the simulation start time. The simulations specifically explored the behavior of the Leader Election algorithm with increasing packet loss probabilities, ranging from 0 to 1 in increments of .1. Unless indicated, all of the graphs in this section display simulation results.

In Table 5.1, we summarize the metrics tracked in our simulations for the Leader Election algorithm. The parameter *tmax* represents the time until convergence is reached by all processes, *num_msgs* represents the number of messages generated in that time, and *rounds* indicates the number of rounds that transpired. The *avg_rounds* parameter tracks the average number of rounds until convergence, as each process may converge in a different round. Convergence is the point in time when all processes agree that they have elected as leader the process with the largest identifier.

Metric	*Description*
tmax	time until convergence
num_msgs	number messages sent until convergence
rounds	number rounds until convergence
avg_rounds	average number rounds until convergence

Table 5.1: **Convergence Metrics Simulated.**

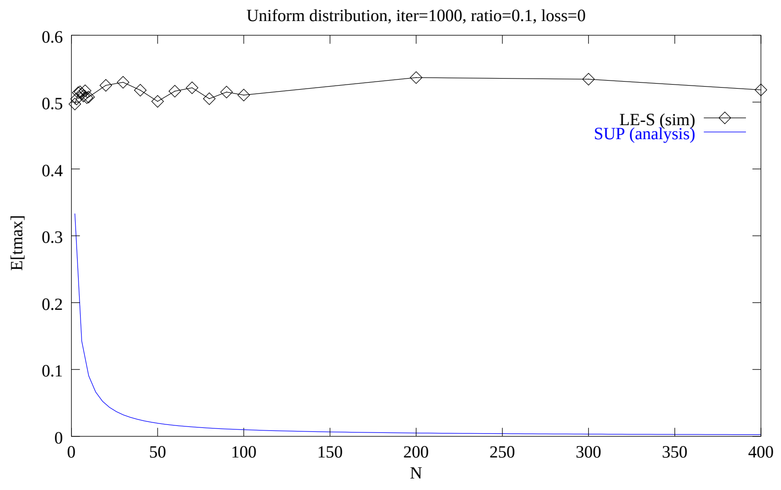

Figure 5.6: **LE vs. SUP:** $E[t_{max}]$ **vs.** N **(**$l = 0$**).**

5.4.1 Comparison with Suppression

In Figure 5.6, we display the delay until convergence, comparing the performance of LE-S simulations with that of SUP analysis in the lossless case (which matched SUP simulations that were shown in Chapter 2). Both the LE and SUP algorithms assume $T_S = T_A = 1$ and $\Delta = .1$, use a uniform timer distribution function for the selection of wake-up times, and examine the lossless case.

It is important to note that convergence has a slightly different meaning under the Suppression algorithm than it does under the Leader Election algorithm. Convergence for SUP measures the earliest halting time of the algorithm, the point in time when all processes agree that they are suppressed. We refer to this point in time as *tmax* because it is equivalent to the last time a message is sent in Suppression, t_{max}, as discussed in Chapter 3 and shown in Figure 3.1. Thus, the convergence delays in the graphs exclude Δ, i.e., they show the maximum time at which the last required message was sent not received. In addition, the Suppression algorithm never goes beyond one iteration because the receipt of any message, including ones own, is sufficient to cause "convergence." This has considerable impact on the time for convergence as compared to LE, as we show below. Nonetheless, convergence marks the completion time of the algorithm in both cases.

Convergence time for LE-S is equivalent to the metric for leadership delay $E[leadership\ delay\ LE-S]_{l=0} = E[t_s] + \Delta$. As can be seen from the plot of LE-S, when we use a uniform distribution for

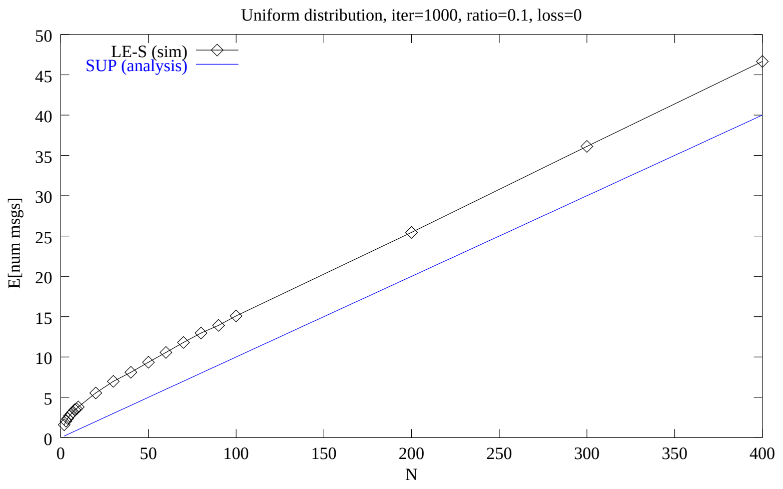

Figure 5.7: **LE vs. SUP:** $E[num\ msgs]$ **vs.** N **(**$l = 0$**).**

the selection of Suppression wake-up times, the delay until leadership is established is the mean of the distribution $E[t_s]$ (plus Δ if transmission delay were included in the graphs). On average the process with the largest process id will choose a wake-up time that is the mean of the distribution because every time is equally likely to be chosen by the leader. In this example, we have chosen a timer interval T_S bounded by 1 unit of time, and indeed the delay is .5 on average. The simulation matches the analysis.

The results for lossless LE-S are similar to the results for Suppression with uncorrelated loss. As the loss probability approaches 1, the average time at which nodes are suppressed ($avg\ t_{max}$, Section 3.8.1) approaches the mean of the distribution. This occurs because, when all N messages are dropped in the network, each node reverts to using its own wake-up time for the the time at which it becomes suppressed, or in this case the time at which it is considered converged.

Interestingly, the delay until convergence is independent of the number of participating processes. The explanation is that we are simply tracking the arrival of a message from the process with the largest identifier. With no loss, N has no impact on the receipt; only the arrival time of the process with the highest id ($E[t_s]$) and the transmission delay (Δ) of the message to the receiver processes will have any bearing on the result.

Figure 5.7 displays a comparison of LE-S and SUP in terms of the number of messages generated until convergence. As expected, Leader Election (LE-S) takes more time to converge than Suppression and generates more messages in doing so. This can be explained by the fact that suppression of a process in LE-S is no longer accomplished by the mere receipt of a message; the message must be from a process with a larger process identifier. This phenomenon is only exacerbated when the loss probability is non-zero. Therefore, convergence may take multiple iterations due to the loss of announcement messages from processes with large process ids, and in turn additional messages may

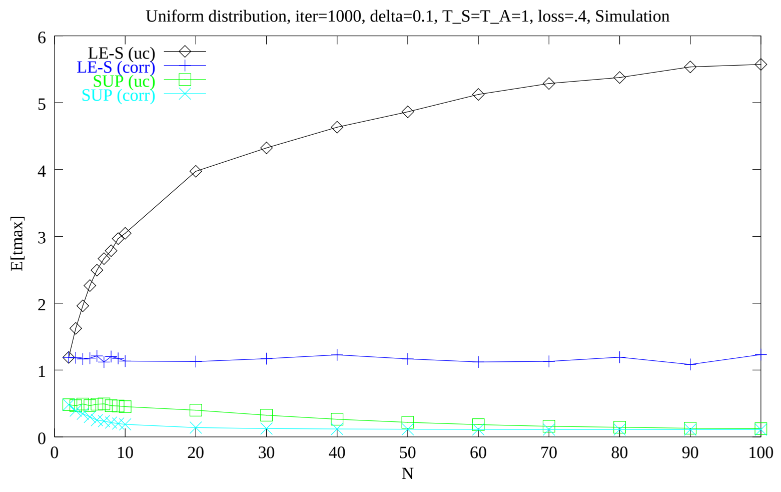

Figure 5.8: **LE vs. SUP:** $E[t_{max}]$ **vs.** N **(**$l = .4$**).**

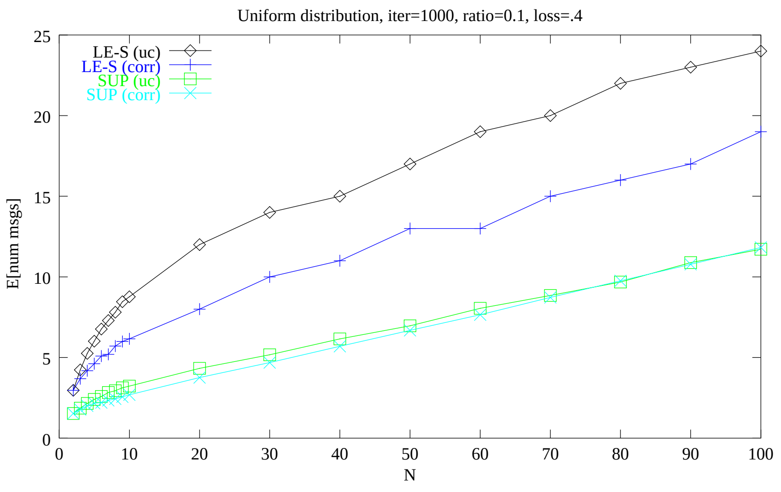

Figure 5.9: **LE vs. SUP:** $E[num\ msgs]$ **vs.** N **(**$l = .4$**).**

be generated.

Next we examine the behavior of SUP and LE under lossy conditions. We use the same parameters in our experiments as earlier, but now explore a loss probability of $l = .4$. Although this loss probability is high, it enables us to amplify the effects of loss and therefore to examine them more easily. In Figures 5.8 and 5.9, we see that loss takes a considerable toll on the performance of Leader Election, especially in the case of uncorrelated loss, and especially compared to Suppression. Increased loss of messages causes an increase in the number of iterations needed for convergence in LE-S, which in turn means the passage of more time and more messages before leadership is established.

In the case of correlated loss, the analysis for $E[leadership\ delay]_{corr}$ shows that convergence

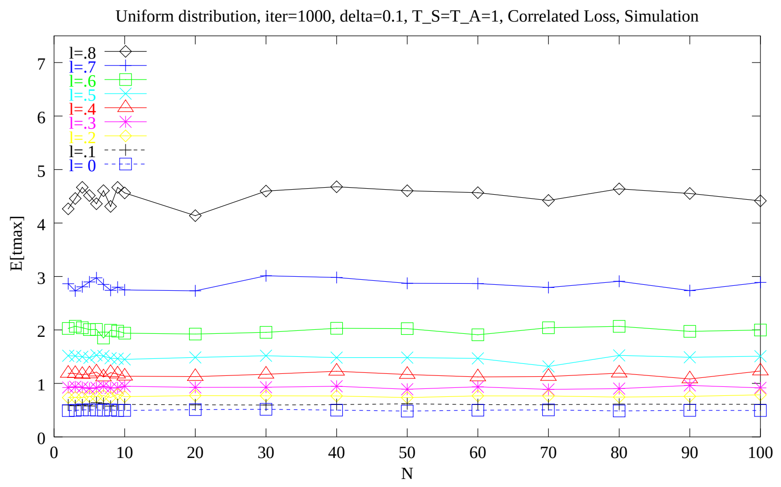

Figure 5.10: $E[t_{max}]$ **vs.** N**: Correlated Loss (Varying** l**).**

time is independent of group size N, as was the behavior in the lossless case (Figure 5.10). For example, given $T_A = 1$ and $E[t_s] = .5$, the plots reveal that the simulations match the analysis (excluding Δ, due to the definition of *tmax*). The results for $E[leadership\ delay]_{uc}$, the convergence delay for uncorrelated loss, however, are a function of group size, but in the limit (as N becomes large) these results are asymptotic as well.

The LE algorithm exhibits the same best and worst case performance as SUP. For both convergence time and number of messages, uncorrelated loss gives upper bounds for the expected behavior of the algorithm.

5.4.2 Comparison with Announce-Listen

A comparison of LE and AL shows that, for small N, LE behaves similarly in performance to AL for convergence time, in that the shape of the curves is similar (Figure 5.11). LE is like AL in that it effectively sends periodic announcement messages and converges when all processes receive the leader's announcement. In AL, convergence corresponds to the point in time when all processes are consistent with regard to the state they store for the leader. The difference between the operation of the two algorithms is the point at which announcements begin. For AL it is immediate, whereas for LE it is at $E[t_s]$. This difference can be seen in the convergence times in the graphs shown in Figure 5.11 (again, convergence time is displayed minus transmission delay Δ, thus showing when the convergence message was sent not received).

Initially the convergence times depicted as $E[tmax]$ along the y-axis show a disparity of $E[t_s]$ between AL and LE. However, note that there is a point at which LE becomes superior to AL. For Figure 5.11, where $l = .4$, that point occurs at $N = 200$ (though the degradation of the performance of AL begins at $N = 70$). This is an example of the need for an adjusted loss parameter, as discussed

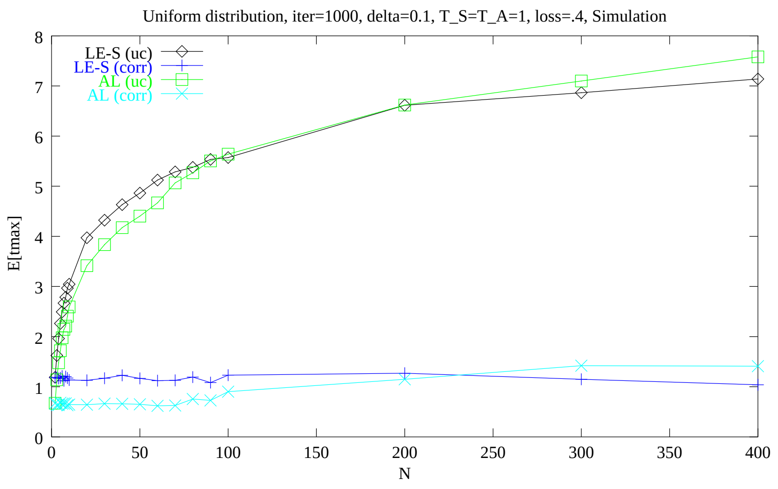

Figure 5.11: **LE vs. AL:** $E[t_{max}]$ **vs.** N **(**$l = .4$**).**

in Chapter 1. This is precisely why LE-S is more attractive than LE-A for multipoint algorithms with large N; it exhibits better scaling properties for large groups.

We do not compare the number of messages generated for LE and AL because they have quite different meanings. Under LE the number of messages generated is the number of processes that believe they are leaders accumulated over the number of rounds they hold this belief. For AL, all processes generate messages with every round.

5.4.3 Leader Selection Criteria

The analysis of the performance metric for leadership delay is quite straightforward. It is the average time it takes for the leader process to awaken in the Suppression interval, denoted $E[t_s]$, plus the time it takes for the leader's announcement to reach the other processes; $E[leadership\ delay]_{l=0} = E[t_s] + \Delta$.

Nonetheless from this simple formula, we can see that a leader selection policy based on largest process address, a traditional approach that appears in countless LE algorithms, will not perform as well as one that selects a leader based on the minimum wake-up time the process chooses in the Suppression interval. The problem boils down to the fact that the mapping between largest process id and Suppression wake-up time is purely random, which explains why, when the Suppression timer distribution function is uniform, on average the process with the largest process id will choose a wake-up time that is the mean of the distribution.

However, we know from our analysis of SUP that the leader selected for its minimal wake-up time leads to an $E[t_s]$ term that is equal to the minimum delay metric in SUP, and thus outperforms the more common approach to leader selection. In effect, this new approach creates the optimal mapping between leader selection criterion and timer wake-up function.

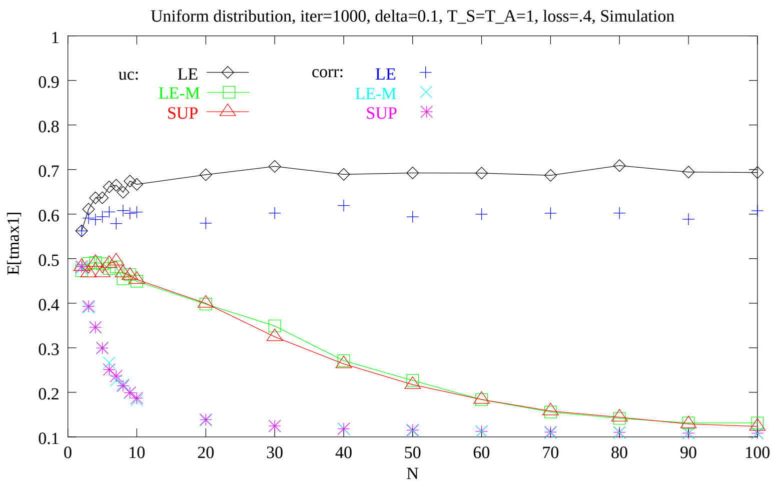

Figure 5.12: **First Round Comparison of LE-M with LE and SUP:** $E[t_{max}]$ **vs.** N **(**$l = .4$**).**

Therefore, for optimal convergence properties (minimal time to converge and fewest number of messages generated), $\vec{Q}$ and $\vec{T}_{min}$ would have to map to each other perfectly. In (Chapter 3), the vector $\vec{Q}$ was defined as an ordering of largest to smallest process identifiers, and the vector $\vec{T}$ was an ordering of wake-up times from earliest to latest. The process with the largest address would need to select the earliest wake-up time within the Suppression interval, the next largest address would need to select the next earliest time, and so forth: q_0 selects t_{min_0}, q_1 selects t_{min_1}, ...

However, here we suggest that minimal leadership delay occurs when $\vec{Q} = \vec{T}_{min}$. In other words, instead of using largest process id as the basis for leadership selection use the minimum wake-up time. Or alternatively, we can think of $\vec{Q} = \vec{T}_{min}$ as suggesting the need for a more direct mapping between **any** leader selection criterion (e.g., largest process id, smallest process load, greatest connectivity) and the Suppression timer wake-up function. For example, in the examples used throughout the chapter, make wake-up time a function of the process id. Processes with larger addresses probabilistically would acquire smaller wake-up values.

We compared the leadership delay for SUP and LE with a modified LE which used the minimum wake-up time as the leader selection criterion and which we refer to as LE-M. When an optimal mapping occurs, LE-M achieves its lower bounds and behaves identically to SUP, at least with regard to the first round of the algorithm. In Figure 5.12 we can see that the results for LE-M in the first round closely overlap with the results for SUP. Note that LE-M also achieves its lower bound when there is no message loss, and when LE, LE-M and SUP all complete in one round.

In Figure 5.13 we display the behavior of the three algorithms when there exists a loss probability of .4. The graphs clearly show that LE-M outperforms an LE algorithm using the standard leader policy. Because increased loss leads to multiple iterations of the announce-listen phase of the algorithm, SUP outperforms LE-M, but still serves as a lower bound for optimal performance

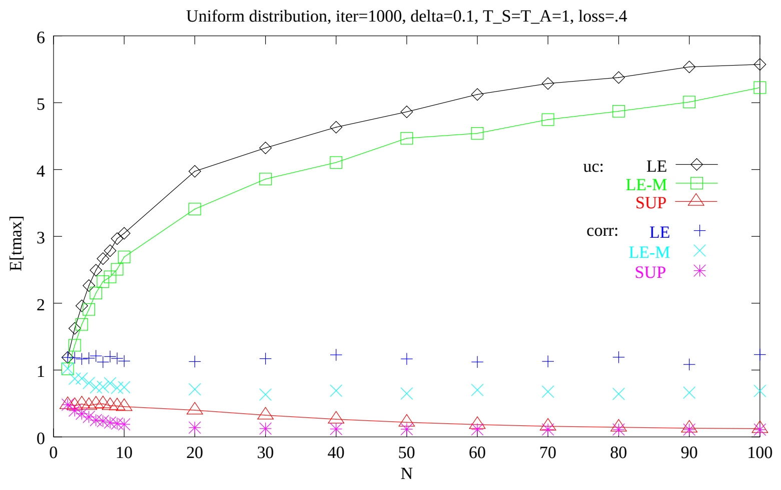

Figure 5.13: **LE-M Compared with LE and SUP: $E[t_{max}]$ vs. N ($l = .4$).**

overall.

The number of rounds raises an important point. We know from the recurrence relation in Section 5.3.3.3 that the number of messages generated in one round impacts the number of messages generated in the next round. Therefore, the fewer messages that are generated in the first round, the fewer messages overall.

We also know from the discussion of timer distribution functions in Suppression (Chapter 2) and from Nonnenmacher's studies [46] that exponential distribution functions lead to fewer messages under certain conditions. Therefore, for applications that are sensitive to messaging overhead, exponential distribution functions for T_S should be considered for the Suppression phase in the Leader Election algorithm.

5.4.4 General Trends

There are several general trends that appear in the metrics. For leadership delay, the performance of LE-S as compared to LE-A, not surprisingly, requires a simple adjustment of $E[t_s]$ units of time. When the leader departs from the system, the re-establishment delay is the same across the no loss, correlated loss and uncorrelated cases. When a late joiner arrives that will become the eventual leader, the re-establishment delay is the same as the leadership delay. If the late joiner is not destined to become the leader, we find that the re-establishment delay for the LE-S case is identical to that for LE-A. This phenomenon occurs because synchronization of a non-leader process with the rest of the processes is independent of the wake-up timer selected by the non-leader process. Re-synchronization is purely a function of how long it takes for the arriving process to hear from the existing leader process.

When moving from a lossless to lossy model, the metric for leadership delay is slightly modified to account for lost announcements, and the delay until all processes converge on a single leader is longer. How much longer is a function of the expected number of announcements beyond the first, given a particular loss level. This can be seen when we compare the simple metric for leadership delay, $E[leadership\ delay]_{l=0} = E[t_s] + \Delta$, with its lossy counterparts, which have additional terms.

When moving from a correlated to an uncorrelated loss model, a metric with a term that uses $E[\#\ consecutive\ lost]_{corr}$ is replaced by one using $E[\#\ consecutive\ lost]_{uc}$; this can be observed in a comparison of $E[leadership\ delay]_{corr}$ and $E[leadership\ delay]_{uc}$.

In the discussion on re-establishment delay, and the ensuing discussion on inconsistent state, we show analytically that these metrics are very much a function of process arrival and departure rates. However, they are also a function of other parameters, notably the timer values T_L and T_A, especially in the case of false departures. False detection of leaderlessness can be avoided on average, by setting $T_L \geq T_A$ in the lossless case, and by setting $T_L > T_A \times E[\#\ consecutive\ lost]$ in the lossy case. We found the same characterization of T_L in AL; the algorithm should be parameterized so that $T_L = kT_A$, where k is a small integer value that should be set according to the number of attempts required for an announcement to reach all participating processes.

We also can observe in the LE metrics and know from our analysis of SUP, that $T_A \geq T_S + \Delta$. If not, then a leader that selects a late wake-up time may not have time to make an announcement before other processes begin making regular announcements, reducing the effectiveness of the Suppression interval.

5.5 Related Work

In the seminal work of Fischer and Lynch [26], it is shown that consensus is impossible in an asynchronous system of processes when there exist unreliable processes. It is for this reason that we examine alternate methods for consensus building in a distributed system prone to failures.

In [56], Singh et al. study the election of qualified leaders. The decision is based on group voting protocols rather than process id numbers. This leads to better selection of leaders because an election can take performance considerations into account. The election schemes that they propose are important for making optimal choices, and one could apply them to the algorithms we have presented in this thesis. However, the actual technique they use for conducting the election is inappropriate in our context because it requires that each node submit a vote that is subsequently tallied and combined with other votes. Their work explores the idea of resiliency by making use of the group majority to reduce the impact of errant group members. We subscribe to a similar philosophy, achieving resiliency in our approach by removing any single-point of failure.

Cidon and Mokryn [16] address the problem of LE in a broadcast environment, taking the traditional distributed algorithm and placing it in a new context. Although quite similar in spirit to our work, we take the additional step of moving the analysis out of the LAN broadcast arena and generalizing the problem further still. By studying the algorithm in a WAN setting, our model diverges from theirs, in that we do not place constraints on communicating processes having to be neighbors, nor do we assume that the communication medium provides reliable or ordered delivery of messages. They assume a concurrent broadcast environment with no losses. Another difference is that they study the case where all processes issue acknowledgment messages to terminate message propagation and the algorithm itself. In contrast, we are free from any specific topology constraints, and only require that participating processes are members of the same group addresses, regardless of how geographically separated the processes may be. We also study the impact of a probabilistic loss model on the operation of LE in a broadcast environment.

Afek and Gafni [2] derive bounds for the performance of LE in both asynchronous and synchronous complete networks, where each process is directly connected to all other processes. Peleg [48] re-examine these results in the context of networks of diameter at most D. This bears resemblance to our approach, in that we assume a multicast group abstraction is provided by a lower layer in the network and means that all nodes are reachable from all other nodes within some maximum transmission delay, in our case Δ. We also provide loss analysis as part of our analysis, and combine Suppression with the initial propagation of leadership messages, in order to reduce messaging overhead.

Fetzer and Cristian [25] study the highly available local leader election problem, which is targetted at fail-aware systems that are prone to group partitions. Like the LE algorithms we study, their algorithm makes use of heartbeat messages for purposes of robustness. However, they design their algorithm for timed asynchronous systems, where delays associated with messages are assumed to be finite but unbounded.

5.6 Summary of Results

In this chapter, we explored two forms of the Leader Election algorithm. We presented the basic approach that establishes leadership immediately by employing Announce-Listen as a first step, and thus was referred to as LE-A. We discussed a refined approach that delays leadership slightly by inserting a phase of Suppression before beginning AL, and as a result has better scaling properties; we named it LE-S. We focused on the analysis for LE-S because LE-A can be derived from it by setting the initial Suppression interval $T_S = 0$. Although this technique is used by many protocols, LE-S had not been analyzed formally.

To evaluate the performance of LE, we explored several metrics: Leadership Delay, the delay until leadership is established at the outset of the algorithm; Re-establishment Delay, the delay until leadership is re-established when leadership is perturbed, as might happen when the leader departs, processes falsely detect leaderlessness, or when new processes arrive into the system; The Number of Messages Generated, and; Inconsistent State, the fraction of the time that the algorithm leads to ambiguity about leadership.

We discussed these metrics under several loss scenarios: no loss, correlated loss and uncorrelated loss. Where appropriate, we modelled the performance of these metrics when all processes arrived into the system simultaneously, during steady state, and when there were perturbations in the leadership.

We compared the results of these metrics with the bounds we had already established for Suppression (Chapters 2 and 3) and Announce-Listen (Chapters 4) in earlier chapters. We found that the impact on SUP of correlated and uncorrelated loss carries over to LE; the loss model leading to the best or worst case for SUP are the best or worst case for LE-S. We also discovered that leadership delay is independent of the group size N, at least for the lossless and correlated loss cases, and in the limit exhibits asymptotic behavior for the uncorrelated case.

Compared to AL, LE-S takes longer to converge for small N, and the convergence is shifted by the mean delay of the Suppression interval $E[t_s]$. As group size grows, however, the performance of LE-S is superior to AL (and LE-A for that matter) due to simultaneous messages and results in a larger effective loss probability for AL.

Not surprisingly, Leader Election generates more messages than the Suppression algorithm due to the differences in the way that participating processes become suppressed. The need to receive a message with a higher process id takes significantly longer than the time it takes for the mere receipt of another message. In addition, Suppression completes in one round, whereas LE continues for multiple rounds, meaning LE takes longer to stabilize. The increased delay incurred by Leader Election translates into increased message generation.

We showed that the traditional approach to leader selection that is based on greatest process id performs sub-optimally for protocols that incorporate the LE-S algorithm. This is because the mapping of process id to wake-up timer value is random. We discovered that a leader selection criterion based on minimal wake-up time was an improvement over the standard method. Furthermore, we suggest that making wake-up time a function of the property governing leadership would be an improvement to standard practice.

5.7 Future Work

Reducing Re-establishment Delay. As the listen timer is set to T_L, it may take a considerable period of time for a participant in the LE algorithm to detect that the leader has departed. For multicast applications that require timely detection of this event, applications have resorted to the use of explicit `Bye` messages, which are announcements containing departure state information rather than arrival state information. The idea behind this action is to decrease the delay before process leadership departure is detected.

We would like to study the impact of such an announcement on LE. If we know the level of loss in the system, then we can calculate how many Bye messages to send on departure. If the departing process is a non-leader, then announcement of departure is unnecessary.[3] If the departing process is the leader, then a number of messages ($E[\textit{announcement i establishes leadership}]$ messages) must be generated to reach all members. If leaders depart frequently, this could have a substantial impact on the numbers of messages generated by the algorithm.

Other Definitions of Leadership. In the early examples throughout this Chapter, we assumed that no mapping existed from $\vec{Q}$ and $\vec{T}_{min}$, the vector of process addresses ordered from greatest to smallest and the vector of selected wake-up times ordered from earliest to latest, respectively. Later we showed that the performance metric for leadership delay can be improved when we set $\vec{Q} = \vec{T}_{min}$. In other words, LE exhibited optimal performance when we replaced the conventional definition of a leader (the process with the greatest identifier) with one based on the minimal Suppression wake-time time.

The idea of an improved mapping follows naturally from what Singh et al. [56] propose in their work. They suggest that instead of using the highest process id to elect the leader (which randomly chooses a leader), use a value that reflects a property that the application is trying to maximize (or minimize). There are several alternatives that make sense for the types of applications that are supported by our loosely-based communication model. For data consistency, a process could announce an estimate of the level of loss it has observed (e.g., a sum of the losses experienced over a given time interval, across all group members). For optimal placement of a leader within the group, use the sum of the delays observed between nodes, selecting the leader with the lowest value.

However, these alternate definitions bring us back to the issue of providing a good mapping between the property to optimize and the timer selection function for LE-S. In the future, we would like to investigate these and other definitions that would lead to "good" leaders [56]. In particular, we want to understand the effect of linking new definitions for leadership with the wake-up timer function, in effect combining the performance needs of the application with the inherent structure of the LE-S algorithm.

Relatedly, what is the impact of changing the algorithm to support multiple agreed-upon leaders? It is common practice that unicast versions of DNS require two copies of each registry to exist for reliability. With the arrival of multicast extensions to DNS [58] [21], it may fall within the province of LE-S to supply multiple leaders for registry sharing. In addition, cluster computing, client-server, and load-balancing applications also may require multiple leaders for redundancy and fault tolerance.

[3]For protocols, such as RTP[54][51] that actually track or approximate the level of group membership, Bye messages are useful from any group member, leader or not.

Composition and Parameterization of Existing Algorithms. We would like to examine other instances, besides LE, where SUP is used in combination with AL. This is part of the larger goal to understand if when multicast components are used in composition, whether or not their metrics can be used compositionally. We have found here that SUP and AL provide good lower bounds for LE-A and LE-S, but there is enough dissimilarity that the metrics do not compose.

Would there be utility in creating an LE algorithm based on SUP followed by AL (SUP+AL), without modification to SUP? That is, elect a leader in LE with the simple receipt of a message, as in SUP itself? This algorithm would not guarantee a single leader, as multiple nodes may elect themselves. Nor would it guarantee the presence of any leader, as the algorithm has the potential to deadlock, e.g., due to message loss or asymmetric delays, process A hears from B, who hears from C, who hears from A. Each defers to a different process to send leadership announcements, and none actively becomes the leader.

However, SUP+AL might be useful in other contexts, not only to avoid message implosion at initialization (the standard reason to precede AL by SUP), but also to limit the number of announcements sent by AL. For example, the algorithm could be used only to allow a random number of announcers to announce in each AL interval. The decision to announce or not would be based on the time a process had selected inside the Suppression interval; basically, the suppression values would be used to randomize the set of announcers allowed to send in each round. This technique could be used in lieu of adapting the announce timer T_A to keep the bandwidth usage below a particular threshold. This type of modification brings the SUP+AL algorithm closer to the individual components, and hence closer to the metrics for SUP and AL.

There are existing protocols that use SUP+AL, but SUP is used here only to spread out the announcements in time, rather than suppress most or even some of them [8]. We know from the analysis of SUP in Chapters 2 and 3, when the earliest and latest messages will be sent on average: $tmin = tmin_0$, and $tmax = tmin_{n-1}$ respectively. The latter is not normally the case, but is here because all messages are generated, thus the last message sent selected the largest t_{min}. Because SUP is merely used to separate the announcements in time, we can trivially gauge the numbers of messages required for convergence: N.

Extensions to LE. There are a few other forms of Leader Election that exist and that have metrics no doubt related to the variations of LE algorithms we examined in the thesis. They introduce additional opportunities to study the relationship between timer intervals. We would like to determine when these variations are improvements upon LE-A or LE-S.

1. **Query.** An explicit request or `Query` message is sent at the outset of the algorithm to find the leader, followed by an announcement if no leaders respond. If the Announce timer duration, T_A, is long compared to the response time needed by the application, a Query can trigger a

reply sooner than the next announcement. Note that a `Query` functions like an announcement message, in that it shares leadership information among processes. However, it generates a reply from the leader(s).

2. **Suppress, then Query.** Suppression precedes the Query and is used to reduce the number of messages generated should all processes begin at once, or become synchronized. If none are received with a greater address by the selected wake-up time, a process queries to find the leader, announcing leadership if no leaders respond. Note that suppression is also used when responding to a Query. This is an improvement over the Query technique in that it generates fewer messages. However, it introduces greater delay until leadership is established.