Group Leader Election under Link-State Routing

*Yih Huang and Philip K. McKinley**

Department of Computer Science
Michigan State University
East Lansing, Michigan 48824
{huangyih, mckinley}@cps.msu.edu

Abstract

In this work, we place the problem of group leader election in a context "inside the network," meaning that participants in the election process are network switches/routers, rather than hosts. A robust solution to the problem, called the Network Leader Election (NLE) protocol, is proposed for use in networks based on link-state routing (LSR). The protocol is robust, for it achieves leadership consensus in the presence of adverse events, such as leader failures and network partitioning. The correctness of the protocol can be proved formally. A simulation study reveals that the NLE protocol incurs low overhead in handling leader failures and in group creation. In addition, it is shown how important network functions, including hierarchical routing, address resolution, and multicast core management, can benefit from the NLE protocol.

1 Introduction

The problem of leader election concerns the selection of a distinguished member from a set of computing systems that are interconnected by a network. This problem has been extensively studied in the context of distributed computing systems. Generally speaking, solutions to the problem are distributed "host-level" algorithms that make use of various network services, such as reliable delivery of messages, in order to monitor the working status of the established leader or cast ballots for a new leader. Well-known contributions in this area include the Bully algorithm [1] and the Ring algorithm [2].

In this paper, we address the leader election problem as it occurs "inside" the network. The participants in the election process are assumed to be switches (or, interchangeably in this work, routers), rather than hosts or application processes. Solutions to this problem are intended to support underlying network functions, as opposed to being directly invoked by user applications. Whereas a host-level election protocol typically considers the underlying network as a "black box," a *network-level* election protocol can see and take advantage of the internal operation of the network, in particular, the underlying routing protocol.

Network functions that can make use of an efficient leader election protocol are several. In this paper, we focus on three examples. First, in Asynchronous Transfer Mode (ATM) networks and other hierarchical networks, switches in a low-level subnetwork (called a routing domain) select a switch to represent the domain in the next routing level [3]; a solution to this *domain leader election problem* supports routing operations within the network. Second, many address-mapping services, such as the mapping between group addresses and member addresses [4] and the mapping between network addresses and link-layer addresses [5], use a central server approach; a solution to the *server assignment problem* selects a leader to undertake the server responsibilities. Third, some IP multicast protocols, such as CBT [6] and PIM [7], identify a network node, called a core node, as the traffic transit center for each multicast group; a solution to this *multicast core management problem* supports multicast services provided by the network. A common requirement of solutions to the above problems is fault tolerance: since network functions/services are expected to survive not

*This work was supported in part by DOE grant DE-FG02-93ER25167 and by NSF grants CCR-9503838, CDA-9529488, and CDA-9617310.

0-8186-8061-X/97 $10.00 © 1997 IEEE

only single-point failures, but also component failures that partition the network, the solutions to these problems must also survive these adverse scenarios.

Our proposed solution, called the Network Leader Election (NLE) protocol, is based on *link-state routing* (LSR) [8, 9], an increasingly popular type of network routing. Under LSR, the local status of each switch, including its ID and incident links, is learned by the network via the broadcasting, or *flooding*, of link-state advertisements (LSAs). Based on received advertisements, each switch locally maintains a complete topology image of the network, which it uses to make routing decisions. Prominent LSR protocols include the Open Shortest Path First (OSPF) protocol [8] for use in the Internet and the Private Network-to-Network Interface (PNNI) standard [3] for ATM networks.

The proposed NLE protocol extends LSR to include group-leader binding LSAs, which are used by group members to advertise their choice of leader. Upon receiving such an LSA, other switches in the network either accept this selection, or choose and advertise an alternative leader. The objective of the NLE protocol is to achieve network-wide consensus on leader bindings. Moreover, the NLE protocol achieves the following properties in fault tolerance:

1. **[Leadership Consensus Property]** Given a group G and a network that has been partitioned into a set of segments $S_1, S_2, \ldots, S_k$, $k \geq 1$, there will be consensus on the leader binding for G within each segment S_i, and that leader will be an operational switch within the segment.

2. **[Mutual Consensus Property]** By requiring group members to report to the established leader, the NLE protocol ensures that, within each network segment S_i, the leader maintains a member list for the group that includes those, and only those, group members in S_i.

It is to be noted that, when the network is not partitioned, the above consensus properties hold throughout the network. Results of a simulation study show that these features can be achieved with minimum protocol overhead.

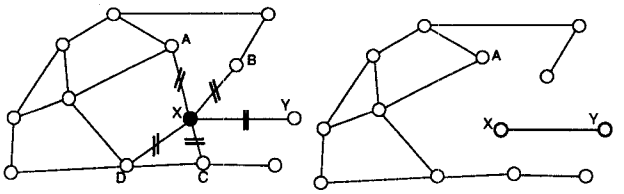
The remainder of this paper is organized as follows. Section 2 reviews the operation of LSR and a previous network-level leader election method, the ATM domain leader election protocol, that satisfies the two consensus properties discussed above. The design of the NLE protocol is presented in Section 3. The performance of the NLE protocol and that of the ATM domain leader election protocol are compared via simulation in Section 4. In Section 5, we discuss the application of the NLE protocol to the address resolution problem and to the multicast core management problem; included are simulation results regarding the performance of NLE in creating multicast groups. Finally, conclusions are given in Section 6.

2 Background

2.1 Link-State Routing

A routing protocol disseminates network information that switches use to find paths for relaying communication traffic. In the case of LSR, complete knowledge of the network is made available to all switches. For this purpose, every switch broadcasts throughout the network its local state, including the ID of the switch, links incident to the switch, bandwidth of individual links, and so forth. Since the network images must be updated in response to network dynamics, every switch constantly monitors its local state and immediately advertises any changes. For example, when a link fails, the value of its working state changes from ON to OFF, producing a *link-down* LSA from each of the two switches connected to that link. Similarly, *link-up* LSAs will be flooded when the link later returns to a functional state.

Although it may be tempting to represent the states of switches in a similar manner to that of links (for example, "switch-up/switch-down" LSAs), such states are not defined in LSR. The reason is that an LSR protocol cannot distinguish a failed node from one that becomes unreachable due to failed links. To illustrate, let us consider the example shown in Figure 1(a). Assume that node X has crashed. The five switches that are neighbors of X (A, B, C, D, and Y) detect the lack of response on the five links incident to X, and subsequently flood five link-down LSAs. The network as perceived by switch A (and any other switch other than X and Y) after the link-down LSAs have been flooded is depicted in Figure 1(b). Switch A will receive only four of the link-down LSAs because switch Y, which advertises the failure of the (X, Y) link, has been isolated from the rest of the network. As shown, the configuration stored at switch A does not reflect the true state of the network. This example illustrates that an LSR protocol cannot determine whether a switch has failed or not. Instead, the protocol should be concerned with "reachability" of the switch. As we shall see, the NLE protocol relies on the underlying LSR protocol to provide such "leader reachability" information that is needed in handling leader failures and network partitions. Likewise, member reachability information is used by the leader to handle member failures.



(a) Node X crashes. (b) The perception of the network at node A.

Figure 1. Indeterministic of node failure.

We emphasize that LSR-based protocols, including the NLE protocol, are not intended for *direct* implementation in very large networks or internets. LSR itself is generally intended for use in a set of networks under one administrative authority (in Internet terminology, an *Autonomous System*) which typically contains a few hundred switches and possibly several thousand hosts. In order to apply such protocols to larger networks, a routing hierarchy can be introduced. The combination of an LSR protocol and a routing hierarchy has been adopted by the ATM PNNI standard [3], as discussed next.

2.2 Domain Leader Election in ATM Networks

ATM is a connection-oriented communication technology that relays small fixed-size cells in hardware. As a telecommunications standard, ATM adopts a hierarchical routing method to cope with the routing complexities of networks that are national or even global in scope. In this approach, a large network is divided into segments, called *routing domains*, and the routing decision is divided into two levels, high-level inter-domain routing and low-level intra-domain routing. Given a source/destination pair, inter-domain routing determines a path comprising a series of domains from the source domain to the destination domain. Within each domain on this path, intra-domain routing determines a path that comprises a series of switches that connect the two domains on either side. In the ATM Private Network-Network Interface (PNNI) standard [3], switches in a domain use LSR to perform intra-domain routing, and elect a leader switch to represent the domain in inter-domain routing.

The domain leader election protocol defined in the current ATM standard [3] is LSR-based. In this protocol, each switch maintains two election-related states: *leader priority* and *preferred leader*. The former is manually configured to determine the rank of the switch. To determine its preferred leader, every switch independently searches its local *domain image*, the topology image of the domain, for the switch that has the highest leader priority among reachable switches within the domain. Like other LSR switch states, values of the two states at all switches within the domain are flooded throughout the domain and constitute parts of the local domain images at switches. If a switch identifies itself as the preferred leader, after waiting for a period of time, it inspects its local domain image for the preferred leaders of the other switches. When unanimity is achieved, the candidate advertises its leader status.

As an example, let us consider a routing domain where the administrator configures a default leader X with leader priority 3 and a backup leader Y with priority 2. The remaining switches are configured with priority 1. We assume that the leadership of X has been established, that is, the current preferred leaders of all switches are X. Now consider what happens when the leader X fails. In this case, neighboring switches of X notice the unresponsiveness of X-incident links and flood link-down LSAs. With these LSAs, every operational switch finds X unreachable, and searches through its local domain image for a switch with the next highest priority. In this example, the result would be Y with priority 2. All switches change their preferred leader to Y and advertise this information. These advertisements could be considered as "ballots," which switch Y must collect before claiming itself the new leader.

The current ATM election protocol can handle leader failures, and if every switch periodically advertises its preferred leader, the protocol achieves the leadership and mutual consensus properties defined earlier However, the protocol is relatively expensive in terms of bandwidth, since every switch must perform flooding, and it is restricted to a fixed leader selection method (namely, its leader ranking scheme). In the NLE protocol, only a small number of leader candidates perform flooding; other members in the group use point-to-point messages, rather than LSAs, to cast ballots. Further, the NLE protocol is designed to operate independently of the leader selection criteria used by different groups, enabling its use as a single election protocol to support a variety of network functions.

3 The NLE Protocol

3.1 Overview

The operation of the NLE protocol is summarized below. Since some decision-making processes of the NLE protocol, such as the leader selection policy, are ap-

plication dependent, we discuss the protocol operation in the context of the domain leader election problem. Adaptation of the protocol to other problems is discussed in Section 5.

1. For every group g, each switch x maintains a *leader binding* $Binding_x(g)$, whose value is a triple $(Leader_x(g), Source_x(g), Stamp_x(g))$, where $Leader_x(g)$ is the leader of the group g as perceived by x, $Source_x(g)$ is the switch that suggested this binding, and $Stamp_x(g)$ is the timestamp associated with the binding. The goal of the NLE protocol is to maintain consensus on $Binding_x(g)$ values across the network.

2. When a switch x joins a group g, it searches for the $Leader_x(g)$ entry in its local database. If the entry is not found, group g is said to be *unbound* at x. In this case, switch x selects a switch c as the leader of the group according to a *leader selection policy*, sets $Leader_x(g)$ to c, and broadcasts this binding. For the domain leader election problem, the leader selection policy selects a reachable switch with the highest leader priority.

3. Once the switch x has a $Leader_x(g)$ entry, it sends a JOIN-REQUEST message to switch $Leader_x(g)$. The join operation will not be considered successful until the return of a JOIN-ACK from the $Leader_x(g)$. Further, the switch x must re-join g (that is, repeat the join process) each time the $Leader_x(g)$ value changes.

4. When a switch x leaves a group g, it sends a QUIT-REQUEST to switch $Leader_x(g)$. Again, the quit process does not not finish until the corresponding QUIT-ACK returns from $Leader_x(g)$.

5. When $Leader_x(g) = x$, switch x acts as the leader of the group g: it processes JOIN-REQUEST/QUIT-REQUEST messages, and returns appropriate acknowledgments. Further, via join and quit requests from members, the leader maintains a member list for g, denoted as $ML_x(g)$. A member of g will be dropped from $ML_x(g)$ if it becomes unreachable from the leader x. We point out that, since members are required to re-join the group each time a new leader is elected, a new member list will be compiled at the new leader. Member lists are not required at switches other than the leader.

6. When a switch x that is a member of a group g finds the switch $Leader_x(g)$ unreachable, switch x selects and broadcasts a new leader binding for g. To avoid a rush of new leader bindings from all members of g, a delay timer of random length is used to postpone the re-selection task. Typically, one member wakes up before others and advertises a new binding. The remaining members simply accept the binding and re-join the group.

7. Even when switch $Leader_x(g)$ is still reachable from x, the switch x may decide, according to application-specific leader performance criteria, to select and advertise a new leader for group g. Given a group g, an *objection policy* determines when a switch objects to the current leader binding and selects a new leader. For the domain leader election problem, a switch objects to the current domain leader when it discovers a reachable switch has a higher leader priority than does the current leader.

8. Each switch periodically advertises a list of groups for which it is the leader. Formally, switch x periodically advertises a list of group IDs, G_x, where a group $g \in G_x$ if and only if $Leader_x(g) = x$.

3.2 State Machines and Events

At a switch x, the NLE protocol defines two finite state machines (FSMs) for each active group g: a Membership Status Machine, denoted as MSM(x, g), and a Leadership Consensus Machine, denoted as LCM(x, g). Both LCM(x, g) and MSM(x, g) machines access the $Binding_x(g)$ entry; such accesses are assumed to be atomic to avoid race conditions. Figure 2 shows the events processed by the two machines. The LCM(x, g) processes incoming leader bindings for the group g, and reacts to events that indicate problems with the current leader, such as leader-unreachable events and objection events defined by the objection policy. The MSM(x, g) handles join and quit events and is responsible for ensuring that the current leader, $Leader_x(g)$, holds correct information regarding the membership status of the switch x. The MSM(x, g) also processes leader-change events, which are raised whenever the LCM(x, g) accepts a new binding for group g.

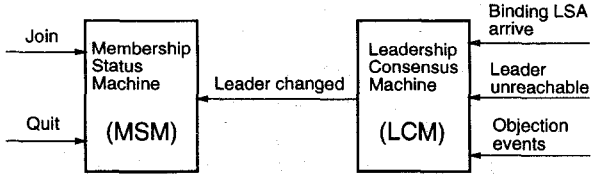


Figure 2. The finite state machines in NLE.

3.3 The Operation of LCM

The state transition diagram for the LCM is depicted in Figure 3. As shown, an LCM comprises four states: EMPTY, PENDING, REMOTE, and LOCAL. The EMPTY state is the initial state of LCMs. When there is no binding regarding g at x, the LCM(x, g) is in the EMPTY state; the values of $Leader_x(g)$ and $Source_x(g)$ are undefined, and the value of the $Stamp_x(g)$ is defined to be zero. An LCM(x, g) is in the LOCAL state when $Leader_x(g) = x$, and in the REMOTE state when $Leader_x(g) \neq x$. The LCM sometimes uses a timer to postpone the task of leader selection. When this happens, the machine enters the PENDING state, waiting for time-out.

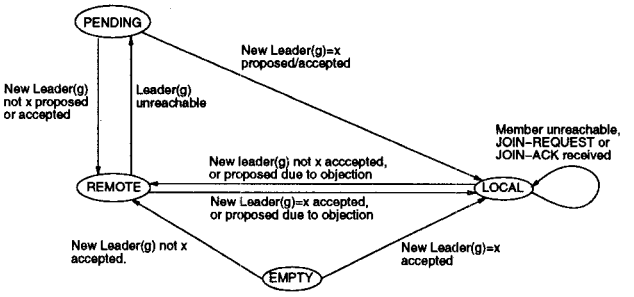


Figure 3. The leadership consensus machine at a switch x for a group g (LCM(x, g)).

A binding LSA is a pair $(g, (c, s, t))$, where the first element g specifies the group and the second element (c, s, t) is the value of this binding. The LCM processes binding LSAs according to the rules below:

A1 An incoming binding LSA $\ell = (g, (c, s, t))$ will be *accepted* at a switch x if $(t, s) > (Stamp_x(g), Source_x(g))$, otherwise it is rejected at x. (The comparison is in lexicographical order.) This rule guarantees that more recent bindings override old ones but that the reverse will not happen. When ℓ is accepted, its value (c, s, t) becomes the value of $Binding_x(g)$. Subsequently, the LCM(x, g) enters either the LOCAL or REMOTE state, depending whether new $Leader_x(g)$ is x or not.

A2 When a switch x proposes and advertises a leader c for a group g it 1) increases the $Stamp_x(g)$ by one, 2) sets $Source_x(g)$ to x and $Leader_x(g)$ to c, and 3) floods a binding LSA $(g, Binding_x(g))$. The LCM(x, g) then enters either the LOCAL or REMOTE state, depending on whether new $Leader_x(g)$ is x or not.

There are two situations where the LCM(x, g) may use Rule A2 to propose and advertise new leader bindings for the group g: when the $Leader_x(g)$ becomes unreachable, and when an objection event is raised according to the objection policy. In the latter case, the LCM proposes a new leader only if the machine is in the REMOTE or LOCAL state. When the current leader of g becomes unreachable from a switch x, the switch is triggered to select and advertise a new leader. To avoid a rush of simultaneous leader binding LSAs from group members, the LCM(x, g) sets up a delay timer and enters the PENDING state. There are two ways for the LCM to leave the PENDING state: 1) the timer fires, and the machine selects/advertises a new leader according to Rule A2, or 2) an "acceptable" binding LSA arrives before time-out. In case 2, the delay timer is canceled, and the LCM processes the LSA according to Rule A1. We will discuss the effects of various timer values in Section 4.

When the LCM(x, g) enters the LOCAL state, switch x must create a member list for group g and process JOIN-REQUEST/QUIT-REQUEST messages from members of g. The member list of g is created every time LCM(x, g) enters the LOCAL state, and is destroyed every time LCM(x, g) leaves that state. JOIN-REQUEST/QUIT-REQUEST messages will be acknowledged and used to update the member list when LCM(x, g) is in the LOCAL state, but are discarded silently when the machine is in any other state. When an unreachability event concerns a switch y that is not the leader of the group g, the action of LCM(x, g) depends on whether x considers itself to be the leader. If so (that is, $x = Leader_x(g)$), x removes y from the member list of g; otherwise, it discards the event.

3.4 The Operation of MSM

The MSM at a switch x for a group g, denoted as MSM(x, g), reacts to join(g) and quit(g) events. An MSM has four states: MEMBER, JOINING, NON-MEMBER, and LEAVING, among which the NON-MEMBER state is the initial state. With respect to a group g, the MSM at a switch x is in JOINING state if it wishes to join the group but has not completed the "registration" procedure, namely, the exchange of JOIN-REQUEST and JOIN-ACK messages with the leader, $Leader_x(g)$. After the JOIN-ACK message is received, the joining member enters the MEMBER state. Defined similarly, a member of g is in the LEAVING state during the exchange of QUIT-REQUEST and QUIT-ACK messages with the leader, and will enter the NON-MEMBER state after completion. Retransmissions of REQUEST messages may be necessary to ensure successful delivery.

The MSM does not deal directly with the leader-

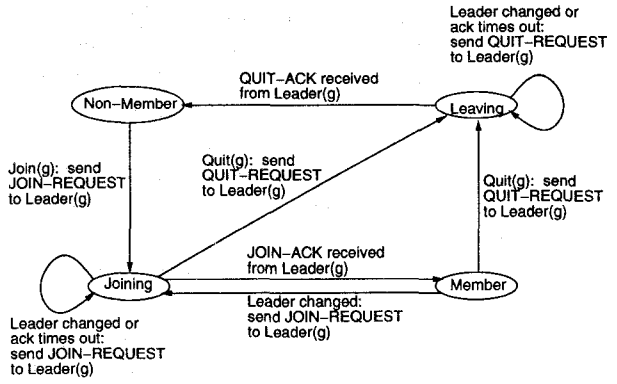


Figure 4. The membership status machine at a switch x for a group g (MSM(x, g)).

unreachable events. However, when the LCM(x, g) changes the leader binding due to leader-unreachable events or other objection events, it generates a leader-change event to be handled by the MSM(x, g). If the switch is a member of the group g, the switch must re-join the group, that is, the MSM(x, g) machine enters the JOINING state so that JOIN-REQUEST and JOIN-ACK messages are exchanged with the new leader.

If a switch x joins a group g when the LCM(x, g) is in the EMPTY state, the switch must select and advertise a leader for the group, following the procedure defined in Rule A2. For every switch v in the network, including LCM(x, g), this advertisement will be received by LCM(v, g).

3.5 Proof of Correctness

The correctness of the NLE protocol, specifically, that it exhibits the leadership consensus property and the mutual consensus property, can be proved formally. Details are omitted here due to space limitations, but can be found in a companion technical report [10].

4 Performance Evaluation

In this section, we investigate the performance of the NLE protocol in handling leader failures. Specifically, the NLE protocol is compared against the ATM domain leader election protocol [3]. In our simulations, networks comprising up to 400 switches were used. For each network size, 40 graphs were generated randomly, and two simulation sessions were conducted on each graph. Table 1 shows the characteristics of the graphs generated. In the table, the symbol T_f denotes the worst-case time to perform a flooding operation in a given network. The following flooding protocol is assumed: LSAs arriving at a switch for the first time are forwarded along all incident links, except the incoming one. LSAs arriving at a switch for the second time are dropped silently. LSAs are forwarded to neighboring switches one by one. For each LSA forwarding, we used software overheads as measured on the ATM testbed in our laboratory. The testbed comprises Sun SPARC-10 workstations equipped with Fore SBA-200 adapters and connected by three Fore ASX-100 switches. From these measurements, we obtained the figure 600 μsec, which includes the overhead at both the sending and receiving switches.

Network size	Avg. degree	Avg. diameter	T_f (in ms)
10	3.6	3.25	3.56
20	3.57	4.75	5.12
40	3.73	6.18	6.71
60	3.88	6.8	7.5
80	3.91	7.08	7.87
100	4.12	7.15	8.1
120	4.10	7.53	8.42
140	4.20	7.73	8.64
160	4.22	7.58	8.8
180	4.31	7.75	8.96
200	4.29	7.95	9.08
250	4.50	7.95	9.34
300	4.70	7.98	9.47
350	4.87	7.85	9.53
400	5.07	7.85	9.62

Table 1. Characteristics of randomly generated graphs.

We consider two metrics for the performance of leader election: the leader-binding convergence time and the number of leader binding LSAs produced for an election. The former refers to the length of the period from the moment the election begins to the moment that all network switches agree on the same leader node. (When an election is held due to the failure of the current leader, the election begins at the moment the leader fails.) The latter measures the number of leader-binding LSAs that are sent before consensus on the leader node is reached. In addition, we measured the bandwidth consumption of the two approaches. This is motivated by the fact that switches use point-to-point messages to cast ballots in the NLE protocol, but must use flooding operations in the ATM election protocol.

As discussed, the NLE protocol uses a random timer to delay the flooding of new leader binding LSAs when the current leader fails. In this study, we assumed that the current leader crashes at time 0, and that delay

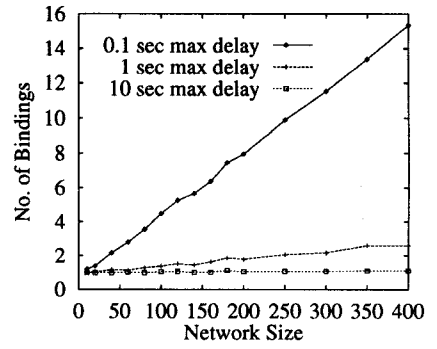
timers are uniformly distributed between 0 and a simulation parameter *max_delay*. We used max_delay values of 0.1 seconds, 1 seconds, and 10 seconds.

The results regarding the metric of the number of bindings are plotted in Figure 5(a). Even the very short maximum delay value (0.1 seconds) introduces fewer than 16 bindings in 400-switch networks. When the maximum delay is set to 1 second, fewer than 3 bindings are generated in large networks. When the maximum delay value of 10 seconds is used, only one binding is created in almost all simulation sessions. Although not shown in the figure, the current ATM election protocol produces N preferred-leader LSAs, the equivalent of binding LSAs, in an N-switch network for every leader failure event.

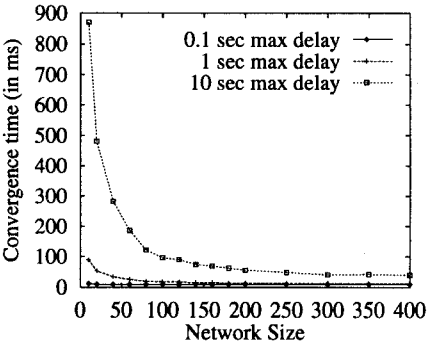
The results for convergence time are plotted in Figure 5(b). For this performance metric, the shorter the maximum delay value, the faster the convergence, since a short maximum delay value produces early time-out of the delay timers, and hence switches take less time to flood new leader bindings. Also, the larger the network size, the faster the binding converge; not surprisingly, a large number of switches that set up random delay timers tends to produce one that times out quickly.

The results for bandwidth consumption are plotted in Figures 6(a) and (b). Bandwidth consumption is measured by counting the total number of links traversed by every LSA and JOIN-REQUEST/ACK messages associated with the election. Figure 6(a) shows the results of the NLE protocol, which uses flooding operations to broadcast leader bindings and point-to-point messages to cast ballots (that is, to send JOIN-REQUEST messages). The curves in Figure 6(a) conform with those in Figure 5(a), that is, the more concurrent bindings produced, the more bandwidth consumed. The bandwidth consumption of the ATM election protocol is significantly larger than that of the NLE protocol, as shown in Figure 6(b).

In summary, compared to the ATM leader election protocol, the NLE protocol incurs far fewer flooding operations and consumes a small fraction of the bandwidth. We further emphasize that the ATM leader election protocol requires every switch to periodically advertise its preferred leader, while the NLE protocol requires only the leader to periodically broadcast its leader status. We conclude that the NLE protocol is more efficient than the ATM leader election protocol, while being equally robust.



(a) number of bindings.



(b) convergence time.

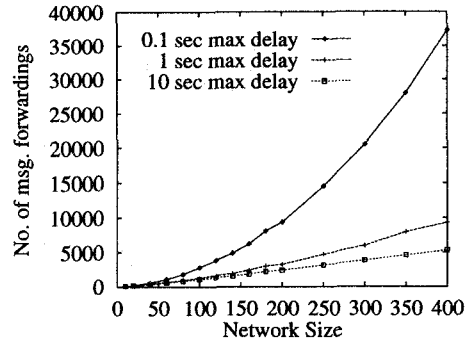
Figure 5. Performance of the NLE protocol.

5 Other Potential Uses of The NLE Protocol

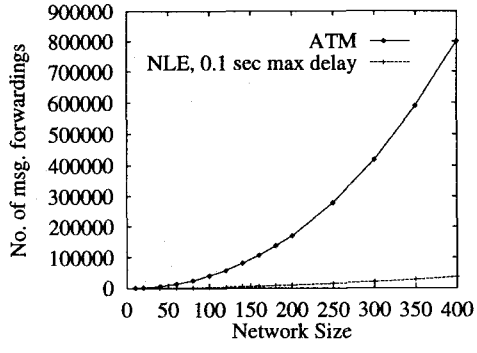
5.1 Multicast Address Resolution

In the last several years, a great deal of research has addressed the issue of implementing IP over new link layer protocols, such as ATM/AAL5. One of the difficult tasks in implementing IP over ATM networks is how to handle multicast addressing. Whereas IP allows a source node to send a datagram to an abstract multicast group address, the current ATM standard does not support such an abstraction. Rather, ATM supports multicasting through point-to-multipoint unidirectional virtual channels, which require the sender to explicitly establish a connection to each destination.

One approach to this problem is to use a Multicast Address Resolution Server (MARS) [4], a central server that acts as a registry, associating IP multicast group identifiers with the ATM interfaces representing the members of the groups. The MARS is queried when an IP multicast address needs to be resolved, and hosts and routers must update the MARS when they join and leave groups. As a centralized solution, however, the potential for MARS failure is an important issue.



(a) NLE bandwidth.



(b) ATM bandwidth.

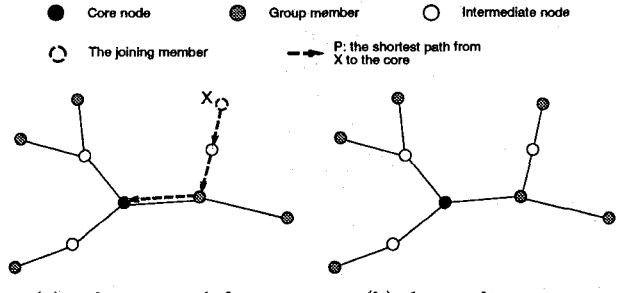
Figure 6. Bandwidth usage of alternative election protocols.

The approach described in [4] is to manually configure nodes with the addresses of one or more backup MARS nodes that they can contact in descending order of preference.

An alternative method is to use an election protocol, such as NLE, to "automatically" handle MARS failures and, just as important, accommodate network partitions. Such an implementation might work as follows. A specific group identifier (call it MARS-GID) is reserved for the election of the MARS; every switch is assumed to be a member of this group. The selection and objection policies of the MARS follow a ranking scheme similar to those for domain leader election. If the current MARS crashes, the NLE protocol is used to establish consensus on a new $Leader_x$(MARS-GID) binding. For the new MARS to operate properly, member lists must be re-collected. To this end, every switch in the network maintains an *interested multicast addresses* (IMA) list, $M_x = \{m_1, m_2, \ldots, m_k\}$, where each element m_i is a multicast address that one or more of the attached hosts is interested in. This list is usually maintained by a local membership management protocol, such as the IGMP [11]. Since every switch must send a JOIN-REQUEST to a newly elected MARS, reconstruction of member lists at the MARS can be implemented by augmenting each JOIN-REQUEST message from switch x to include a copy of M_x. The consensus properties of the NLE protocol guarantee that, should the network be partitioned, there will be a MARS within each segment that maintains multicast group member lists for those, and only those, switches in the segment.

5.2 Multicast Core Management

The Internet supports *multicast communication*, the delivery of a message from a single source to multiple destinations, and *group addressing*, whereby a set of hosts can be referred to by a multicast group address. Some prominent IP multicast protocols, such as CBT [6] and PIM [7], associate a multicast traffic transit center, or *core node*, with each multicast group. In such approaches, datagrams destined to a multicast group are first forwarded to the core node, from which they are distributed along a multicast tree to reach group members. A new member joins the group by sending a JOIN-REQUEST message toward the core node. The path traversed by the message defines a tree branch to reach the new member. An example of this join process is shown in Figure 7. Figure 7(a) shows the shortest path P from a joining member X to the core node. The result of this join operation is shown in Figure 7(b).



(a) a shortest path from the joining member to the core.

(b) the resultant tree.

Figure 7. Member join operation in the CBT/PIM protocols.

We point out that CBT/PIM JOIN-REQUEST messages are not issued by host members. Rather, when a host joins a group, it is its *ingress switch*, the point at which the host is attached to the network, that sends the request on behalf of the host. That switch consequently becomes a *switch member* of the multicast group, and need not send further JOIN-REQUEST messages when other attached hosts sub-

sequently join the group. In the following discussion, the term member refers to switch member.

The association of the core node with a multicast group can be modeled as a leader election problem, and the NLE protocol can be applied. One approach is as follows. We assume that a core election is held whenever a multicast group is created (that is, when the first member joins), and that a new core is elected if the current core fails. Regarding the core/leader selection policy, we can assume that the default is the *random member* policy [12]: whenever a member is required to select and advertise a core node (including group creation time), the member simply recommends itself. A number of other core selection policies are discussed in [12] and could be incorporated into the NLE protocol. As with the applications discussed earlier, the mutual consensus property of the NLE protocol enables arbitrary multicast groups to handle network partitions and re-unifications.

The maintenance of the leader binding of every active group at every switch in a network may raise the concern of scalability. In [13], we describe another core-management method that addresses this issue by using the NLE protocol to select a central server to maintain the leader bindings of all active groups. The method presented above, however, has an advantage in group-join time, because joining switches do not have to query a server to resolve leader-group bindings. This feature is important to situations where members of a group join and exit at a high rate.

5.3 Performance of Multicast Group Creation

When the NLE protocol is used in domain leader election and MARS election, all switches in the network are members of the (single) group, and the group is assumed to be created at network initialization, a relatively rare event. In the case of multicast core management, on the other hand, there are many multicast groups directly tied to applications. More importantly, group creation time may be important to the performance of those applications.

Therefore, we conducted a study to evaluate the performance of the NLE protocol when a multicast group is created. As in the previous performance study, we are interested in two performance metrics: convergence time and the number of binding LSAs. It turns out that the convergence time in this case is quite predictable, as shown in the following theorem.

Theorem 1 *Given the flooding diameter T_f of a network (the worst-case time to finish a flooding operation), the convergence time for group creation under NLE is less then $2T_f$, assuming that no network component failures occur during group creation.*

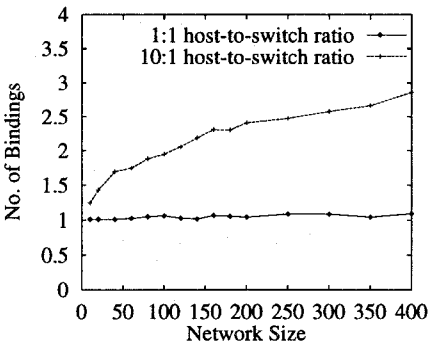
Proof: See [10]. □

To investigate the number of binding LSAs produced by group creation, we simulated the creation periods of multicast sessions with M participants. The arrival time of each participant is normally distributed with mean zero, the predetermined startup time of the group. The standard deviation value is set in such a way that 99% of the participants arrive within a predetermined time interval; this interval will simply be called an *arrival interval*. We used arrival intervals of lengths 1 second and 0.1 seconds. A switch joins the multicast group when its first attached participating host arrives. In a simulation session, the size of the participant population, M, is controlled by the participant-to-switch ratio; we used the values of 1 and 10 in this investigation. As such, our simulation study covers a wide range of participant population sizes, from 10 (obtained by 10-switch networks with 1 participant per switch) to 4000 (obtained by 400-switch networks with 10 participants per switch). Simulation sessions involving a small number of participants could represent teleconferencing applications, whereas those involving very large population sizes may represent Distributed Interactive Simulation (DIS) applications. The combination of very short arrival intervals with very large population sizes produces extremely busy group creation periods, in order to stress the NLE protocol.

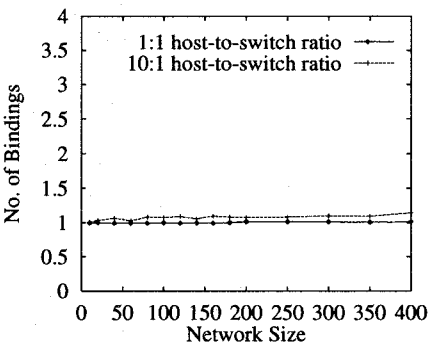
Figure 8 shows the results of this study. Figure 8(a) plots the results when using the 0.1 seconds arrival interval. The worst case in the figure is only 3.0, meaning that even when 4000 participants join a multicast group within 0.1 seconds, the NLE protocol produces only three leader binding LSAs. Figure 8(b) plots the results for the 1 second arrival interval. As shown, this relatively longer (but still very short) arrival interval produces virtually no redundant bindings, that is, there is one leader binding produced per group creation event. We believe that the results in Figure 8 demonstrate that the NLE protocol is a viable method for handling many real-world situations.

6 Conclusion

We have considered a long-established distributed computing problem in a novel context. Specifically, the leader election problem has been studied in a context where participants of the election process are network switches and applications of the problem are



(a) 0.1 seconds arrival intervals.



(b) 1 second arrival intervals.

Figure 8. Number of bindings generated for group creation.

network functions, including, among others, hierarchical routing, address resolution, and multicast communication. The proposed solution, called the Network Leader Election protocol, models the group-leader binding problem as a consensus problem under link state routing. In this model, the local network images at switches are extended with leader binding entries, whose network-wide consistency is guaranteed by the protocol. The correctness of the NLE protocol, including its leadership consensus property and the mutual consensus property, can be proved formally. Our simulation studies reveal that the NLE protocol incurs minimal overheads for multicast group creation and moderate overheads to handle leader failures. The performance of the NLE protocol compares favorably with a previous network group leader election protocol for ATM networks.

Further Information

A number of related papers and technical reports of the Communications Research Group at Michigan State University are available via anonymous ftp and world-wide web. The respective addresses are `ftp://ftp.cps.msu.edu/pub/crg` and `http://www.cps.msu.edu/~mckinley/crgweb`.

References

[1] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Trans. on Computers*, vol. 31, pp. 48–59, January 1982.

[2] N. Fredrickson and N. Lynch, "Electing a leader in a synchronous ring," *Journal of the ACM*, vol. 34, pp. 98–115, January 1987.

[3] ATM Forum, "Private network-network interface specification version 1.0." ATM Forum technical specification af-pnni-0055.0000, March 1996.

[4] G. Armitage, "Support for multicast over UNI 3.0/3.1 based ATM networks." Internet RFC 2022, November 1996.

[5] M. Laubach, "Classical IP and ARP over ATM." Internet RFC 1577, January 1994.

[6] A. Ballardie, "Core based trees (CBT) multicast routing architecture." Internet draft, March 1997.

[7] S. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "The PIM architecture for wide-area multicast routing," *IEEE/ACM Trans. on Networking*, vol. 4, pp. 153–162, April 1996.

[8] J. Moy, "OSPF version 2." Internet RFC 1583, March 1994.

[9] J. M. McQuillan, I. Richer, and E. C. Rosen, "The new routing algorithm for the ARPANET," *IEEE Transactions on Communications*, pp. 711–719, MAY 1980.

[10] Y. Huang and P. K. McKinley, "Group leader election under link-state routing," Tech. Rep. MSU-CPS-97-14, Department of Computer Science, Michigan State University, East Lansing, Michigan, April 1997.

[11] S. Deering, "Host extensions for IP multicasting." Internet RFC 1112, August 1989.

[12] K. L. Calvert, E. W. Zegura, and M. J. Donahoo, "Core selection methods for multicast routing," in *Proceedings of IEEE ICCCN '95*, (Las Vegas, Nevada), 1995.

[13] Y. Huang and P. K. McKinley, "LCM: A multicast core management protocol for link-state routing networks," Tech. Rep. MSU-CPS-97-30, Department of Computer Science, Michigan State University, East Lansing, Michigan, August 1997.