Group Communication under Link-State Routing

By

Yih Huang

A Dissertation

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

Department of Computer Science

March 18, 1998

ABSTRACT

GROUP COMMUNICATION UNDER LINK-STATE ROUTING

By

Yih Huang

Multiparty communication, also termed *group communication*, is a generalization of the traditional *point-to-point* communication in which more than two parties can participate in a "conversation." Many current and emerging communication applications, such as teleconferencing, computer-supported cooperative work, and distributed interactive simulation, typically involve several, or a large number of, participants, and require efficient network support for multiparty communication. *Link-state routing* (LSR) is a type of network routing method that makes complete network status information available throughout the network. Adopted by both the Internet, the de facto standard for data communications, and *asynchronous transfer mode* (ATM), an international standard for telecommunications, the importance of LSR in communication cannot be overstated. In this research, we investigate and exploit the relationship between LSR and group communication. Specifically, we develop a collection of novel and efficient protocols that (1) use group communication methods to improve the performance of LSR operation, (2) take advantage of LSR to provide new network services to group communication applications, and (3) benefit both LSR and group-based applications. Our contributions can be summarized in the following four areas.

First, we identify an important aspect of LSR operation that can benefit from group communication methods: the broadcast of network status information, also known as the *flooding* operation. We propose a novel flooding approach for use in ATM networks, termed *switch-aided flooding* (SAF), that takes advantage of underlying

ATM hardware functionality. The SAF method is shown, through both theoretical analysis and simulation study, to be much more efficient than previous methods.

Second, we address a requirement raised by the diversity of multiparty communication applications: the need to support different types of *multipoint connections* (MCs), the network entities that define the routing of traffic streams among the participants in multiparty conversations. We develop a *generic MC* (GMC) protocol that is able to accommodate multiple topology types and computation algorithms as plug-in components. We show that a "chassis" for MC protocols can operate efficiently under LSR.

Third, we investigate an issue involved in both LSR and group communication — the leader election problem. We define the problem of "network-level" leader election, where participants of an election are network switching elements rather than hosts, and we develop an LSR-based solution to the problem, called the Network-level Leader Election (NLE) protocol. The NLE protocol is formally proven to be robust; it handles not only leader failures, but also much more disastrous situations, such as network partitioning. We apply the NLE protocol to the problem of managing traffic transit centers, or *core nodes*, for multicast groups. Our proposed solution, called the LSR-based Core Management (LCM) protocol, automatically selects the core node for a multicast group when the group is created, supports core migration to improve multicast performance during the lifetime of the group, handles the failures of both multicast cores and the core management server itself, and survives network partitioning scenarios.

Lastly, we turn again to the operation and performance of LSR itself. Traditionally, LSR uses two costly techniques to achieve its robustness and responsiveness: message forwarding on every communication link in the flooding of network status updates, and the periodic flooding of local status by each router. We conclude this research by combining two techniques developed earlier, namely the election of a leader and the construction of multipoint connections, to develop a totally different approach to LSR. The resulting *Tree-based LSR* (T-LSR) protocol imposes only a small fraction of the overhead of previous LSR methods, while guaranteeing to main-

tain consistent routing decisions throughout the network under any combination of network component failures, partitioning scenarios, and undetected communication transmission errors. Unlike the ATM-oriented SAF protocols, the T-LSR protocol is designed for use in general-purpose, LSR-based networking environments and requires no special hardware support.

In summary, this research reveals a mutually beneficial relationship between group communication and LSR: many aspects of group communication (such as the construction of communication channels, the management of membership, and the consensus on leadership) can take advantage of the internal operation of LSR, while the performance of LSR itself can be improved by incorporating various group communication mechanisms.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

Many modern distributed applications involve multiparty communication, in which two or more participants are involved in a group "conversation." A distinguishing characteristic of multiparty communication is the requirement for a source party (for example, a person that is currently speaking in a teleconference) to be heard by more than one receiving parties (for example, the other participants in the conference). Applications that involve multi-party communication include teleconferencing, computer-supported cooperative work, distributed virtual reality, remote teaching, tele-gaming, replicated file servers, parallel database search, and distributed parallel processing. This thesis concerns efficient network support for various aspects of multiparty communication, or, interchangeably, *group communication.*

Previous prominent works in this direction exist in the form of multicast protocols, especially those proposed for the Internet [1]. A multicast protocol routes communication traffic streams from their sources to multiple destinations, as opposed to exactly one destination, as in conventional point-to-point routing. Multicast methods supported within the network are generally favored over host-level multicast methods, where typically a source explicitly sends a copy of the message individually to each recipient. The problem with the latter approach is that, when the paths from the source to destinations share a common link, the message traverses the link multiple times. Network-supported multicast methods avoid this redundancy by having the network replicate the message after its traversal of the common link. Representative

IP multicast protocols include PIM [2], CBT [3, 4], DVMRP [5], and MOSPF [6]. An important concept supported/used by such protocols is *group addressing*, whereby more than one communication party can be referred to as a single entity. For example, IP multicast addresses [7], which are perhaps the most well-known group addressing mechanism, allow a data packet that is tagged with a single destination address to be delivered to all the systems that are "listening" to that address.

Most group communication implementations must deal with two issues: the collection and management of group membership information, and the routing of traffic streams to reach group members. Alternative approaches to the former issue range from no network support (that is, no membership management in the network), to maintaining a member list at every network node for every active group. The latter issue concerns the topology computation of *multipoint connections* (MCs), that is, sets of communication channels that connect group members. Various methods of MC topology computation have been devised by researchers to meet different performance criteria, such as the transmission delay experienced by group members and the total bandwidth consumed by the group [8, 9, 10]. Many multicast protocols can be considered as distributed implementations of one, or a small set of, MC topology computation algorithms.

Some group communication implementations must deal with a third issue, namely, *group leadership*, which arises when a multicast protocol assigns special duties to one group member. The leader of a group may serve as the center for membership management, or as a transit point through which all traffic streams destined to the group must be forwarded. Group leaders can be configured manually or can be selected automatically by the network. Important multicast protocols that introduce such "distinguished" members include PIM [2] and CBT [3, 4].

It should be noted that the use of group communication is not restricted to applications; many aspects of the operation of the network itself involve group communication. An important example is the underlying (unicast) routing protocol, a protocol that compiles knowledge of the network for the purpose of making routing decisions. A communication network consists of three major components: hosts, switches (or, synonymously, routers)[1], and communication links. The hosts are computers or other devices that allow users to access the network, while switches relay traffic streams through the network over communication links. When requested to relay a traffic stream toward a given destination, a switch must determine on which of its incident links to send the traffic. To ensure the correctness and quality of this decision, the switch requires knowledge about the rest of the network. One approach to achieve this goal is to disseminate the status and configurations of switches and links throughout the network so that a global picture of the network can be compiled at every switch. As such, the routing protocol uses broadcast operations, a special case of multicast operations in which all network nodes are recipients. In this scenario, the entire network can be considered as a group to which switch status information is sent. Routing in communication networks has been extensively studied in computer science. Although not all routing methods use broadcast operations in this manner, a very important one does.

Link-state routing (LSR) [11, 12] is an increasingly popular type of unicast routing. An LSR protocol makes complete knowledge of the network available to all switches in the manner described above. The local status of each switch, including the bandwidth available at incident links, buffer capacity, and the workload, is learned by the network via the broadcast, or *flooding*, of *link-state advertisements* (LSAs). Based on received advertisements, each switch locally maintains a complete image of the network, which it uses to make routing decisions. The Open Shortest Path First (OSPF) protocol [11], introduced by the Internet community, is one of the most well-known LSR unicast protocols. LSR has also been adopted as the routing method for Asynchronous Transfer Mode (ATM), a telecommunications standard that bases all communication on connection-oriented hardware switching of small, fixed-size cells [13].

This dissertation addresses the interaction between group communication and LSR. Our interest in this problem stems from the following three observations. First,

[1]We will not distinguish the two terms here, despite the fact that one of them may be preferred over the other under certain contexts of discussion.

since LSR involves the maintenance of a complete image of the network at every switch, LSR-based networks might use this information to support a wide range of group communication algorithms. Locally available network images at switches may also help reduce the overhead incurred by distributed implementations of these algorithms. The second major advantage of LSR is its fault tolerance. Because every link is monitored by its incident switches, and every switch is monitored by neighboring switches, malfunctioning components and congested areas are made known to all functioning switches promptly. Even the earliest LSR protocols were able to survive disastrous situations, such as network partitioning [12]. Building group communication facilities upon such a solid foundation has clear implications with respect to robustness. Third, because some important parts of LSR operations exhibit characteristics of group communication, methods targeted at general-purpose group communication may help, or be tailored to help, the LSR protocol itself. As we will demonstrate in later chapters, an efficient group communication method can be used to accelerate the flooding of switch status information, and leader election plays an important role in large-scale, LSR-based networks that are organized in a hierarchical manner. Considering the fact that LSR is being used in the infrastructure of many modern networks, improving its performance will benefit not only multiparty communication applications, but all applications that use such networks.

In this dissertation, we model various aspects of group communication as the *consensus problem under LSR*, which is defined as follows. Due to delays in receiving an event advertisement, switches in an LSR-based network can have different views of the network for a short period of time. The situation is exacerbated when multiple events are advertised simultaneously. Furthermore, a network can be temporarily partitioned due to malfunctioning components, and the resulting subnetworks may evolve independently. The consensus problem under LSR is to guarantee that, given any combination of status changes, component failures, and transmission errors in advertisements, all switches will eventually produce identical images of the network, provided that the network is not permanently partitioned. This definition can be generalized to incorporate group management information, if network images are extended to include such information.

Thesis Statement: *By modeling various aspects of group communications (such as leadership, membership maintenance, and communication channel construction) as consensus problems under LSR, we develop novel and efficient solutions for many important issues of network group communication, including fault-tolerant leader-consensus management, the support of multiple types of multicast communication channels, the handling of disastrous situations, such as network partitioning, and the improvement of the LSR itself.*

The major contributions of this work can be summarized as follows.

1. Switching-aided flooding (SAF). This flooding method takes advantage of ATM hardware cell relaying and duplication to improve the performance of flooding operations in ATM networks. We first develop two SAF protocols, called the Basic SAF and Bandwidth Efficient (BE) SAF protocols, that construct a hardware-based data-distribution tree to accelerate the dissemination of (network-status) information. To further improve efficiency, we develop a third SAF protocol that uses a ring topology to handle acknowledgments efficiently. The complexity of this Efficient and Reliable (ER) SAF protocol is shown to be optimal in terms of bandwidth consumption, workload at switches, and flooding delay. Improving the performance and efficiency of flooding operations can be very important to the responsiveness of the network in meeting diverse application needs.

2. Generic multipoint connection (GMC) protocol. The GMC protocol is based on LSR and can be considered as an MC protocol "chassis," that is, a framework that is able to accommodate multiple existing, and future, MC topology algorithms. Such an MC protocol is expected to benefit a wide variety of multiparty communication applications that favor different performance criteria. For example, a live multimedia broadcast could use an MC topology that minimizes the transmission delays from a single source to a large number of destinations,

while a distributed interactive simulation application may prefer an MC topology that can efficiently accommodate a large number of participants, each of which is both a sender and a receiver.

3. Network-level Leader Election (NLE) protocol. The NLE protocol establishes consistent group-leader bindings at network switches, maintains up-to-date member lists at leaders, and handles network partitioning properly. Specifically, given a group g and a set of network segments $S_1, S_2, \ldots, S_k$, $k \geq 1$, within each segment S_i there will be consensus on a leader for g, and that leader will be an operational switch in S_i that maintains a member list of g containing those and only those members in S_i. The NLE protocol, which is based on LSR, can be used to select traffic transit centers, or *core nodes*, for individual multicast groups and to support hierarchical routing and address mapping. In addition, we apply the NLE in the design of the LSR-based Core Management (LCM) protocol. Rather than conducting leader election on a per-group basis, the LCM protocol uses the NLE protocol to select a switch as the *core management server*, which in turn manages core nodes for all the active groups in the network. Specifically, the LCM protocol automatically selects the core node for a multicast group when the group is created, supports core migration to improve multicast performance during the lifetime of the group, handles the failures of both multicast cores and the core management server itself, and survives network partitioning scenarios.

4. Tree-based LSR (T-LSR). Traditionally, LSR uses two costly techniques to achieve its robustness and responsiveness: message forwarding on every communication link in the flooding of network status updates, and the periodic flooding of local status by each router. We conclude this research by combining two techniques developed earlier, namely the election of a leader and the construction of MCs, to develop a totally different approach to LSR. The resultant T-LSR protocol imposes only a small fraction of the overheads of previous LSR methods, and guarantees to maintain consistent routing decisions throughout

the network under any combination of network component failures, partitioning scenarios, and undetected communication transmission errors. Unlike the SAF work, the T-LSR protocol is designed for use in general-purpose, LSR-based networking platforms, assuming no hardware-based capacities of switches.

The remainder of this dissertation is organized as follows. In Chapter 2, we present background material relevant to this work, including a discussion of the semantics of group communication as perceived by different types of applications, a survey of important multicast protocols, and a survey of link-state routing. We present the SAF protocols in Chapters 3 and 4. Subsequently, we shift our attention to the support of group communication by LSR. The GMC protocol is described in Chapter 5. Chapters 6 and 7, respectively, describe the NLE protocol and its use in the LCM protocol. The T-LSR protocol is presented in Chapter 8. Conclusions and possible future directions are discussed in Chapter 9.

Chapter 2

Background

Advances in communication technology have been dramatic in the last two decades. The Internet, which started out as an experimental project connecting a small number of military sites and universities, has reached all the continents of the Earth. The Internet is no longer a playground for a small group of researchers and academicians, but has become a part of everyday life for millions of people in all kinds of professions. In the meantime, long-established communication infrastructures, such as telephone and cable television networks, are being transformed into modern information superhighways, and are expected to provide a wide spectrum of new services (such as video on demand, multimedia telephony, data communication, tele-gaming, information retrieval, and so forth) directly to individual homes. Moreover, advances in communication technology are not limited to higher bit rates and lower loss rates; they also include unconventional ways of using communication channels. One possibility, which is actively being investigated by many researchers and developers, is to support multiparty communication, whereby more than two communication parties can conduct "conversations." In this chapter, we discuss important multiparty communication applications, existing multicast protocols that support those applications, and link state routing, the type of network routing upon which the proposed methods are based.

2.1 Multiparty Communication Applications

The term multiparty communication, or interchangeably *group communication*, refers to a wide spectrum of communication applications, including human-to-human interaction, distributed interactive simulation, distributed information management, and efficient information distribution. Naturally, such diverse applications have different needs and expectations regarding services provided by the underlying network. Although this dissertation largely concentrates on core network support for group communication, including multicast operations, membership management, and leadership consensus, in this section we examine the applications and services that may be implemented atop such network services. Our objective is to assess and classify the requirements of such applications.

2.1.1 Human-to-Human Interaction

This class of applications brings together individuals for whom it is either difficult or costly to meet face to face (for example, due to their locations), but who must work cooperatively. An example is *videoconferencing*, which allows participants to visually and verbally communicate with others over a network [14, 15]. A special type of teleconferencing, called *computer telephony*, uses computers and data communication networks, rather than public telephone networks, for transmitting audio in real time [16]. Teleconferencing does not necessarily use multimedia; text-based teleconferencing sessions, sometimes called *chat rooms*, have become popular on the Internet [17]. In addition, Computer-Supported Cooperative Workspace (CSCW) applications enable workers who possess different areas of expertise, and who are geographically separated, to remotely and cooperatively conduct difficult operations or manipulate sophisticated equipment [18, 19].

An interesting characteristic of many human interaction applications is the relatively loose requirements on multicast reliability. Typically, these applications can tolerate occasional loss of multicast data at some destinations, since it is human beings, rather than machines, that receive and interpret incoming messages. Occasional losses of characters in a text-based teleconference, for example, may be perceived as typos, rather than transmission errors. When multimedia is used, some loss of image pixels or audio/video frames may produce flares or jumps in playback, but the conversation can continue as long as the degradation is not too severe. On the other hand, delays and jitters in message delivery may be annoying — imagine how to conduct a conversation if one's voice is not heard by others until 30 seconds later. Therefore, many applications in this category use *best effort* multicast, a type of multicast that does not enforce the successful delivery of multicast data at all destinations. When possible, such applications might reserve network resources in advance in order to improve the Quality of Service (QoS) of the network.

2.1.2 Distributed Interactive Simulation

In a DIS application, a virtual environment (VE) is simulated collectively by a set of hosts over a network [20]; examples include a virtual battlefield, a virtual shopping center, and so forth. The interest in DIS originated in the military; a military training session conducted in a virtual battlefield is much less expensive, and more importantly, much safer than a real exercise. Civilian uses of such technology include simulation of police and fire department exercises, as well as the playing of multiparty games across the Internet. In such VEs, some objects are static, such as trees and lakes in a virtual park, whereas other objects are active — they move voluntarily or react to stimuli (people in the virtual park). Some objects may be computer simulated (for example, enemy tanks in a virtual battlefield), while others are controlled by users (for example, tanks controlled by trainees). In general, VE objects must sense and interact with each other in real time. For this purpose, information regarding the current positions, movements, and actions of objects must be disseminated to all participating hosts in a timely manner. Network supported multicast operations and other group communication facilities can be used to improve performance.

DIS applications are often characterized by their scale; the number of participants in a VE can range from a few to thousands, and the underlying network can range from LANs to WANs. The size and the geographic distribution of the participant population raise the concern of scalability issues regarding the underlying group communication support. Moreover, DIS applications call for a special type of reliability, called *selective* reliable multicast [20, 21]. Consider a situation where user X is engaged in a virtual battlefield and unfortunately loses track of his opponent, user Y, due to the loss of a sequence of three messages that broadcast the positions of Y. While the conventional semantics of reliability would force the host of X to request retransmissions of all three messages, X is interested only in the most recent position of Y. A selective multicast protocol ensures the "freshness" of object states maintained at participating hosts, and does not insist on the successful delivery of all state update messages [21].

2.1.3 Distributed Information Management

Single-server solutions have traditionally dominated the area of information management, including the management of file systems and databases. However, for reasons of scalability and fault-tolerance, distributed solutions have been proposed and are gaining momentum. For example, the Coda file system [22] allows for a file system to be replicated at more than one file server. A client to such a file system can retrieve files from the nearest server, but must submit file updates to all servers. Further, servers may fail, and backup systems may join the service. If the client-server communication in such circumstances is modeled as a group communication problem, clients perceive servers as a single network entity, the server group, and should not be concerned with server membership dynamics. Similar methods can be applied to database services, using replicated database servers for either fault tolerance or to improve the performance of query processing through parallel searching.

Many applications in this category demand *atomic* multicast operations, whereby either all destinations of a multicast message receive the message, or none of them receives it. Consider a scenario where a file update request is sent to a group of replicated file servers; an atomic multicast protocol guarantees that either the file is updated at all servers, or at none of them. Although the latter case could be considered as a failed multicast operation, at least it leaves the servers in a consistent state.

2.1.4 Information Distribution

This category refers to applications that disseminate information to a large population. A defining characteristic of such applications is the existence of a single, or a small set of, information sources and a potentially unlimited audience size. For example, in a *remote teaching* application, the lecturer in a virtual classroom can reach a large number of pupils at remote locations.

Some existing information distribution processes can also be re-examined in light of new network technology. For example, the traditional process of distributing public domain software works as follows: the distributor sets up an FTP (File Transfer Protocol) [23] site and interested users individually connect to the site to download a copy. Download requests for popular software may put a heavy load on the FTP server, which repeatedly performs identical tasks: retrieving the software from a local storage medium and shipping it. (It is not uncommon for servers to be brought down by these workloads.) Recently, the HTTP (Hyper-Text Transfer Protocol) [24] and World Wide Web [25] have largely replaced the FTP protocol in this distribution process, but the problem remains. In fact, the situation has become worse due to the more user-friendly interfaces and, hence, a larger number of interested users. A much more efficient approach is to have the distributor (also known as the *publisher*) set up a communication group such that group members, or *subscribers*, simultaneously receive a copy via multicast. *File distribution protocols*, a type of multicast protocols that is designed for this purpose, have been proposed for use in the Internet [26]. File distribution protocols must use *reliable* multicast to ensure the receipt of all multicast data at all destinations. Examples of reliable multicast transport protocols can be found in [27, 28, 29].

2.2 Multicast Communication

Multicast operations, which deliver messages to more than one destination, are centric to the support of multiparty communication applications. The voice and image of a teleconference member must reach all other members. The movements of objects and the status changes of terrain in a VE must be disseminated to all hosts participating in a DIS session. File update requests must be submitted to all servers. And so on. Actually, one may argue that the use of multicast is the defining characteristic of group communication. A *multicast protocol* is a network protocol that defines a set of rules and conventions by which multicast traffic streams are routed from sources to a set of destinations. This section reviews existing multicast solutions developed for two important types of networks, the Internet and ATM networks. We start with a review of routing topologies and membership management techniques.

2.2.1 Multicast Routing Topologies

While many multicast protocols concern simply the construction of individual multicast trees (a set of communication links from a source to a set of destinations), we consider a more general form of multicast routing structure, called a *multipoint connection* (MC), whereby one or more sources can reach one or more destinations. Three major types of MC topologies have been studied:

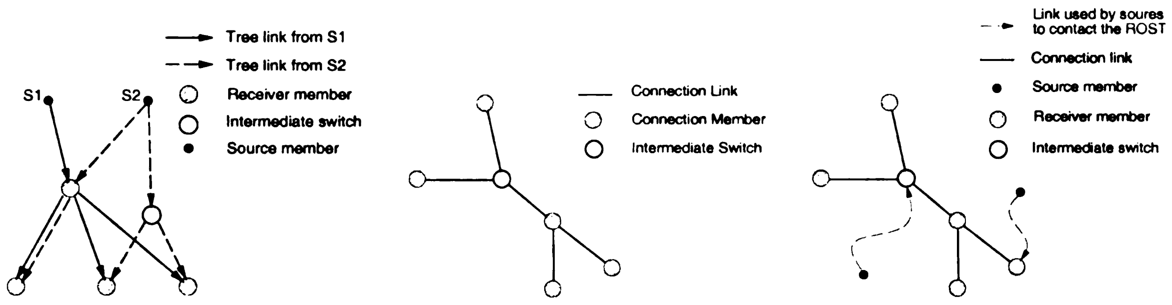
1. *Source-rooted trees* (SRT). The MC topology typically comprises a forest of trees, each individually constructed for a different traffic source. An example in which two trees reach a set of four receivers is shown in Figure 2.1(a). This type of topology is well suited to applications with a small number of senders and a possibly large number of receivers, such as remote teaching and file distribution applications. SRTs are relatively straightforward to construct and are supported by almost all existing multicast protocols. SRT-based MCs are, however, costly to maintain: a new tree must be constructed for each source, and every existing tree must be extended to reach a new receiver. Similar overheads are incurred for departing senders and receivers. SRTs are supported in

the DVMRP protocol [5], MOSPF protocol [6], and the PIM protocol [2], all designed for use in the Internet. The ATM multicast virtual circuit (multicast VC) [30] also supports SRTs.

2. *Symmetric shared trees* (SST). A single tree is constructed to span the members of an MC; every member is both a sender and a receiver (as in the case of teleconferencing). Figure 2.1(b) shows an SST spanning five members. The tree in the figure also uses an *intermediate node* to reach members. Compared with an SRT-based MC, an SST counterpart tends to use fewer network resources (in terms of the number of links) than does an SRT-based forest. The problem of determining an optimal shared tree is the well-know minimum Steiner tree problem [31].

3. *Receiver-only shared tree* (ROST). A single tree spans the receiver members of an MC, while senders use one-to-one unidirectional paths to reach any node on the tree. An example of a ROST with two senders and five receivers is depicted in Figure 2.1(c). The five receivers are connected by a shared tree, depicted with solid lines, and the sender-to-tree paths are represented by dashed lines. This distinction between receivers from senders facilitates membership management on both sides. For example, a group of replicated file servers can be connected by a ROST such that clients to the server group see a single entity, the server MC; individual servers join and leave the server group without disrupting client-to-server communication. ROSTs are supported by the core-based tree (CBT) multicast protocol [32] and the PIM protocol [2, 33].

Besides the type of topology, another issue associated with MC is the topology computation algorithm. Even with a given topology type, different topology computation algorithms can be used, depending on the relative importance of various performance criteria. Such criteria include bounds in transmission delays, network resource consumption, multicast packet loss rate, and so forth. The issue of choosing the right topology algorithm is particularly important to multimedia applications. Such applications typically require quality of service from the network in order to



(a) two SRTs. (b) an SST. (c) a ROST.

Figure 2.1: Three types of MC topologies.

insure the quality of media playbacks. Thus their performance relies on good MC topology decisions so that network components involved in an MC have the capacity and resources to sustain the traffic flowing through the MC. For instance, Zhu [9] presented an algorithm that optimizes cost (for example, bandwidth consumption), in the presence of delay constraints. Bauer [10] examined the multicast tree problem under degree constraints, which may be imposed by hardware switching devices. Waxman [34, 35] addressed the problem of dynamic multicast trees, in which a sequence of membership updates must be carried out one by one. Although this dissertation does not directly address the issue of MC topology computation algorithm, it advocates generic MC protocols that are capable of accommodating a wide range of MC topology types and computation algorithms.

2.2.2 Local Membership Management

In this dissertation, our primary concern is switch/router level multicast. However, from the viewpoint of applications, communication groups are *host groups*; members of such groups are computers or other customer devices that allow users to access networks. Typically, a host accesses the network via a router/switch, called the *ingress* switch of the host, and uses a *local membership management* protocol to inform its ingress switch/router of a list of groups in which the host wishes to participate. The ingress switch maintains a list of groups, where a group is on the list if one or more attached host(s) of the switch is a member of this group. A switch that has at least one attached host that is a member of group G will be referred to as the *switch member* of G; all the switch members of G form a *network group*. With every switch knowing its membership identity with respect to a group, a multicast protocol, when given a multicast message destined to the group, is responsible for the delivery of the message from the *source switch*, the ingress switch of the source of the message, to switch members of the group.

Perhaps the most well known and widely used local membership management protocol is the Internet Group Membership Protocol (IGMP), which is designed for use in broadcast-based LANs [7]. In IGMP, the router of a LAN sends Host-Membership-Query messages destined to a reserved multicast address that includes all hosts in a LAN as members. In response, a host returns a Host-Membership-Report message, which includes a list of multicast addresses in which that host is interested. Via received membership reports, a router compiles a list of multicast addresses in which the network (LAN) is interested. This process is repeated periodically to accommodate membership dynamics. The IGMP uses several optimization techniques to reduce the traffic produced by Host-Membership-Report messages, which must be generated by all hosts in a LAN. Further details of the IGMP can be found in [7].

The group communications solutions developed in this dissertation assume the use of an existing local membership protocol, such as IGMP, by hosts to communicate with respective ingress switches regarding membership identities.

2.2.3 Multicast in the Internet

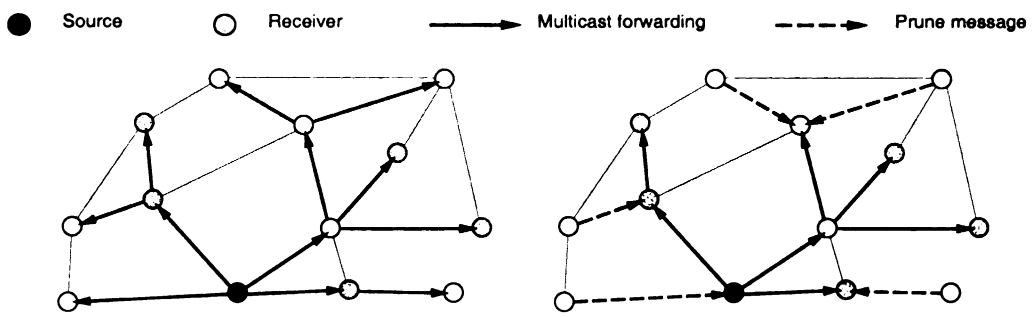
The Internet is a connectionless network, meaning that, when a sender S wishes to send a datagram to a destination D, the sender is not required to contact D prior to transmission. When S and D share a common communication medium (for example, the two are the endpoints of a point-to-point link, or they both have access to a broadcast medium, such as Ethernet), D receives the datagram directly from S. Otherwise, Internet routers collectively deliver the datagram to D as follows: any router R that receives the datagram will forward the datagram via a communication link that constitutes the first hop of an R-to-D shortest path. This forwarding process starts at the ingress router of S, and is repeated until the datagram arrives at D. In this manner, the routing of a given IP datagram is dynamic and independent of other datagrams. The Internet extends this basic point-to-point datagram delivery model with multicast addresses. A datagram that contains a multicast address as its destination is called a multicast datagram, and must be forwarded to all hosts that are interested in the address.

For the discussion of IP multicast, we review four protocols that have been proposed: DVMRP [5], CBT [3, 4], MOSPF [6], and PIM [2]. In this discussion, the term router is preferred over the term switch. Also, the term *multicast group* refers to a set of hosts that are listening to an IP multicast address. Following these semantics, multicast groups in the Internet are *receiver* groups.

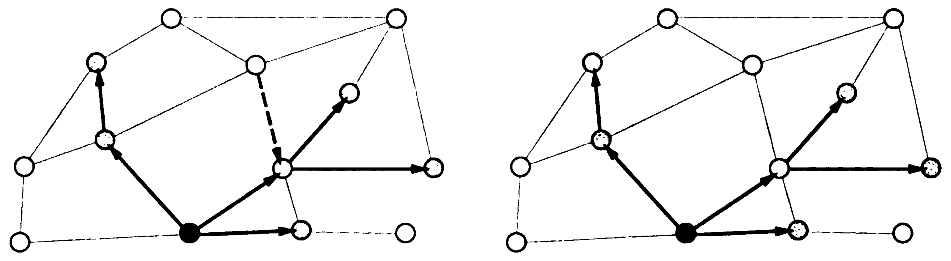
Distance Vector Multicast Routing Protocol (DVMRP)

Given a multicast address M, DVMRP builds an SRT individually for each source of M by means of a broadcast and pruning process. A multicast stream is initially broadcast throughout the network. The broadcast method, called *reverse path forwarding*, works as follows. A router R, upon receiving a multicast packet P that originates from S and is destined to M, determines whether P arrived on a link that constitutes the first hop of an R-to-S shortest path. If so, R forwards P to all neighboring routers except the one from which P arrived. Otherwise, the packet is silently discarded by R. In the meantime, routers that are not interested in M send prune messages "upstream," that is, one hop toward the source S. An upstream router may further discover that all its downstream routers have been pruned from the forwarding tree, and also send a prune message upstream, unless it is itself a member of M. This pruning process will be repeated until all the routers involved in the S-to-M forwarding are either members of M or have downstream members of M, producing an SRT that is rooted at S and reaches members of M.

We use the example shown in Figure 2.2 to illustrate. In Figure 2.2(a), a multicast source is using a broadcast tree to reach five receivers. In Figure 2.2(b), five non-



(a) the broadcast tree. (b) the first step in pruning.



(c) the second step in pruning. (d) the resultant multicast tree.

Figure 2.2: The operation of the DVMRP.

member leaves of the tree send prune messages, which are depicted with dashed lines. In Figure 2.2(c), an intermediate node in the broadcast tree receives prune messages from all its children, and sends a prune message upstream. The multicast tree resulting from this pruning process is depicted in Figure 2.2(d).

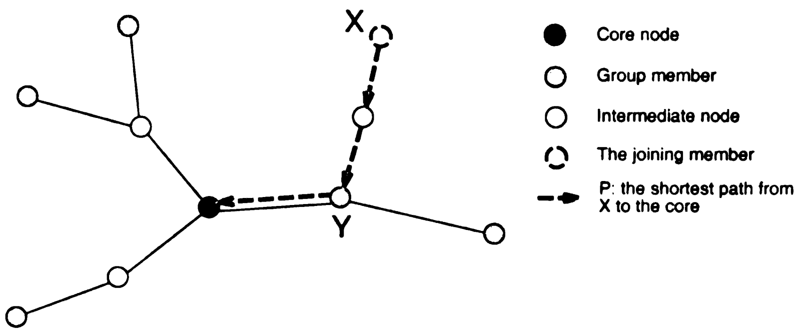
An interesting aspect of the DVMRP is that group membership information is not disseminated, but discovered during tree construction by means of "negative" membership reports, namely the prune messages. However, for this very reason, later membership changes cannot be incorporated into established SRTs. To remedy this problem, existing SRTs must be periodically torn down and re-constructed [5]. This approach causes delays in the handling of membership or network changes. For example, a new member will not receive multicast packets until the next phase of tree re-construction. Periodic tree construction also imposes unnecessary overhead during "quiet" periods, that is, when no changes are taking place. Moreover, shared-tree topologies are not supported by DVMRP. Additional details of DVMRP can be found in [1, 5]. A hierarchical generalization of DVMRP, called Hierarchical DVMRP (HDVMRP), is described in [36].

Core-Based Tree (CBT) Multicast Protocol

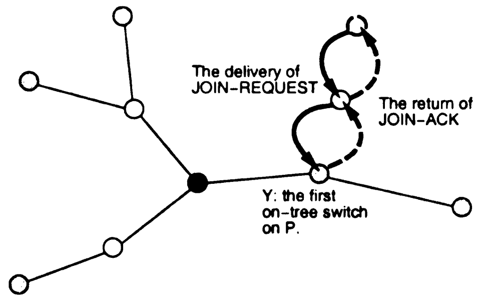
Unlike DVMRP, the CBT protocol [4, 37] builds a shared multicast tree for each group. In the CBT protocol, each multicast group is assigned a distinguished router, called the *core node* of the group. A member joins the group by sending a JOIN-REQUEST message "toward" the core node; the request will stop at the first node that is already on the tree. A branch to the new member is set up by a JOIN-ACK message, which follows the reverse of the path traversed by the JOIN-REQUEST message. A member leaves the group (that is, detaches itself from the tree) by sending a QUIT-REQUEST message to its parent node in the tree, which will also quit if itself is not a group member and has no other children. An example of the member join operation in the CBT protocol is given in Figure 2.3. Figure 2.3(a) shows the shortest path P from a joining member X to the core node. It is switch Y, the first on-tree switch along P, that grants the JOIN-REQUEST and returns a JOIN-ACK message, as depicted in Figure 2.3(b). The result of this join operation is shown in Figure 2.3(c).

The CBT protocol handles adverse network events, including router and link failures, by periodically sending CBT-ECHO-REQUEST messages upstream. If a corresponding CBT-ECHO-REPLY is not heard, a member must rejoin the group by finding another path to reach the core. Compared to the DVMRP protocol, the CBT protocol handles membership changes in an event-driven manner, but still uses a periodic method to incorporate network status changes, causing delays in the handling of such changes. This hybrid approach of handling changes may serve some applications well, but could be inappropriate for critical applications that must operate seamlessly in the presence of network changes.

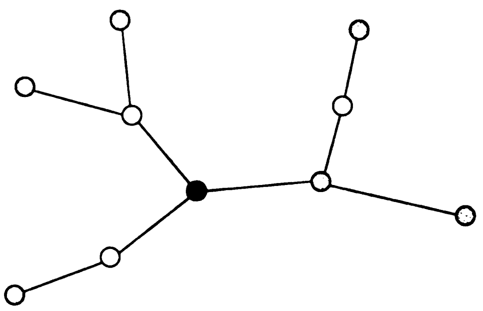
Another concern with the CBT protocol is its inflexibility in MC topology: the protocol does not support the SRT MC topology. Further, the restriction that a multicast packet must be forwarded to the core node before being forwarded along tree



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


The delivery of JOIN-REQUEST
The return of JOIN-ACK
Y: the first on-tree switch on P.

(b) the delivery of CBT messages.

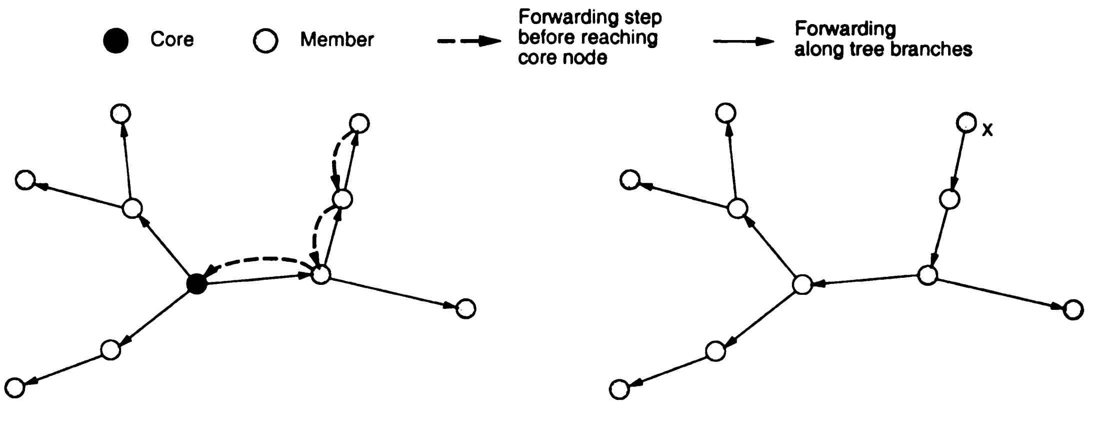


(c) the resultant tree.

Figure 2.3: An example of member join operation in the CBT protocol.

branches imposes unnecessary steps in multicast forwarding. To illustrate the cost of this restriction, let us consider a scenario where group members shown in Figure 2.3(c) are also sources to the group (for example, they are conducting a teleconference). Figure 2.4(a) shows the forwarding of a multicast packet originated from node X, when the session is supported by the CBT protocol. For comparison, Figure 2.4(b) shows the forwarding of the same packet when an SST of the same topology is used. As we can see, the CBT protocol incurs extra forwarding steps, depicted by dashed lines in Figure 2.4, due to its restriction in the starting point of tree distribution.

Besides the CBT protocol, the concept of core based multicast has also been adopted in other IP multicast protocols. Specifically, the Ordered CBT (OCBT) protocol [38] addresses the concern of core failures of the CBT protocol, and the Border Gateway Multicast Protocol (BGMP) [39] constructs core-based multicast


Core
Member
Forwarding step before reaching core node
Forwarding along tree branches
x

(a) using the CBT protocol. (b) using an SST.

Figure 2.4: Comparison of multicast forwarding in the CBT protocol and SSTs.

trees that span across the boundaries of autonomous systems (that is, routing domains in the Internet).

Multicast Extension to OSPF (MOSPF)

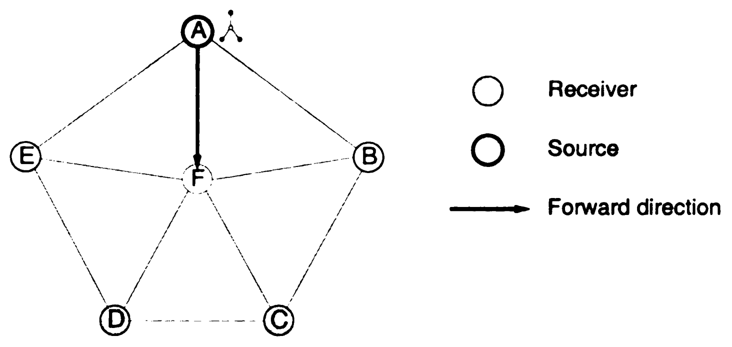
The MOSPF protocol [6] is an extension of the Internet LSR protocol, OSPF [11]. In the MOSPF protocol, the identities of group members are broadcast via *group-membership* LSAs, such that all routers maintain complete member lists for all active multicast addresses. The distribution channel for a multicast group is constructed when the first datagram destined for the multicast address is sent. Upon receiving the first datagram that originates from a source S and is destined for a multicast address M, a router consults its local database for the member list of M and computes a shortest-path tree T that is rooted at the source switch of the datagram, and reaches the switch members of M. Subsequently, the router saves a multicast routing entry such that datagrams from S to M will be forwarded via a set of outgoing links determined by T, and forwards the datagram accordingly. This forwarding will trigger further topology computations at downstream routers.

An example of MOSPF operation is given Figure 2.5, where a host that is attached to router A sends a datagram to a multicast group with members attached to routers C and D. As shown in Figure 2.5(a), router A computes a shortest-path tree that is rooted at A and reaches C and D. This computation is possible because the topology of network is compiled by the underlying LSR protocol, OSPF, while the member list of the destination group ($\{C, D\}$, in this example) is made available by MOSPF. The resultant tree shows that A must forward the datagram to F, which upon receipt will perform the tree computation again and learn of its downstream routers C and D; see Figure 2.5(b). When C and D receive the datagram, they will also carry out the identical tree computation, only to notice that they are leaf routers and should forward the datagram to their attached hosts; see Figure 2.5(c).

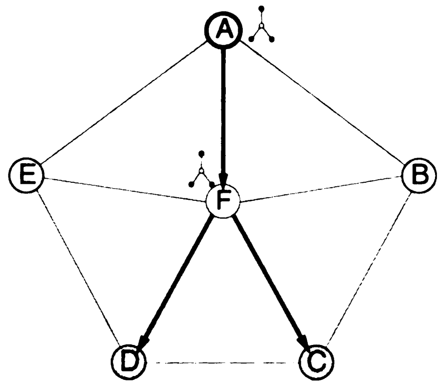
As illustrate, the MOSPF protocol imposes redundancy in topology computation — identical computations are performed at all routers involved in a multicast tree. This problem is exacerbated by the restriction that the MOSPF protocol supports only SRTs; hence this computational redundancy is incurred in a per-source-per-group manner rather than a per-group manner. Furthermore, to adapt to membership and network topology changes after a tree construction process, multicast routing entries created for the tree must be cleared upon the arrival of LSAs that advertise membership or network changes, resulting in the re-construction (and re-computation) of the tree when new multicast datagrams arrive.

Protocol Independent Multicast (PIM)

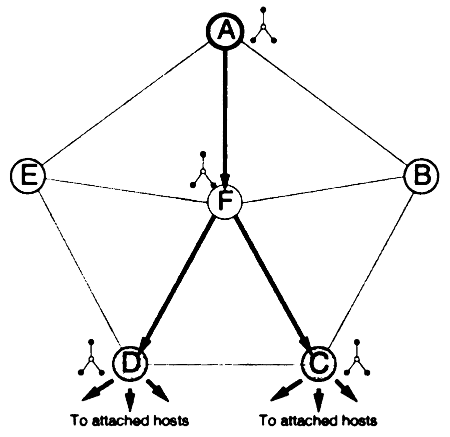
With the MOSPF and DVMRP protocols, every router in a routing domain (or possibly the entire Internet) may be involved in a multicast session. In the case of the MOSPF protocol, every router receives membership change LSAs and maintains member lists for all active multicast groups. With the DVMRP protocol, a multicast stream is periodically broadcast throughout the network. The overhead of network-wide involvement may be justified when a large fraction of the hosts in the network is interested in the multicast; such multicast sessions are sometimes termed *dense mode* multicasts [2]. In contrast, *sparse mode* multicast refers to cases where the participants represent only a small fraction of the hosts in the network and, therefore, network-wide involvement is considered too costly. The PIM protocol supports both dense mode and sparse mode multicast.



(a) The tree computation and forwarding at A.



(b) The tree computation and forwarding at F.



(c) The tree computation and forwarding at C and D.

Figure 2.5: The operation of the MOSPF protocol.

Like PIM, the CBT protocol, which is a representative approach to support (receiver-only) shared-tree MCs, also does not incur network-wide involvement. However, the PIM protocol further emphasizes the need to support other MC topology types, specifically the SRT topology. In addition, the designers of the PIM protocol sought universal applicability of the protocol, and therefore designed the protocol so as not to rely on any specific routing protocol; hence the name Protocol Independent Multicast.

The PIM approach to supporting both dense mode and sparse mode multicast is straightforward; it actually comprises two multicast protocols, one for each mode. In the dense mode, the PIM protocol uses the DVMRP protocol (the MOSPF protocol was not chosen because of its dependence on LSR). For sparse mode multicast, the PIM protocol "initially" builds receiver-only shared trees; the construction of SRTs is performed selectively for some sources during the multicast session. A network region, whether it is a LAN, a routing area, or an Autonomous System, that wishes to participate in a sparse mode multicast, is assigned a *rendezvous point* (RP), which must be a PIM-capable router in that region. The RP of a region plays a role similar to that of the core node in the CBT protocol. Members in that region issue RP-JOIN requests, which serve the same function as the JOIN-REQUEST messages of the CBT protocol, producing within the region a ROST rooted at the RP. If N regions are interested in a multicast address, N different RPs will be associated with the address, and N shared trees will be constructed. The source of a datagram with a given multicast address must forward the datagram to all RPs associated with that address. Each RP will forward the datagram along shared tree branches to reach group members. These concepts are illustrated in Figure 2.6, where two shared trees are constructed for a multicast address that has three sources. Detailed information about the sparse-mode PIM protocol, called PIM-SM, can be found in [33].

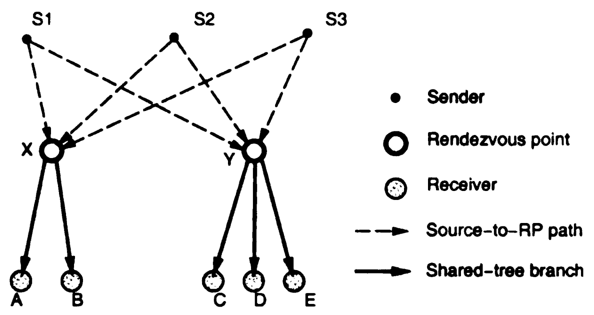


Figure 2.6: Shared trees constructed by the PIM protocol.

The PIM-SM protocol constructs SRTs by means of a topology transition process, which operates in a data-driven manner. When router members of a multicast address observe heavy traffic from a source S, they may determine that the source could be better served by a private distribution channel, and issue SOURCE-JOIN requests to S, resulting in a multicast tree that is rooted at S. Continuing the previous example, Figure 2.7 shows that an SRT has been built for the source S_3.

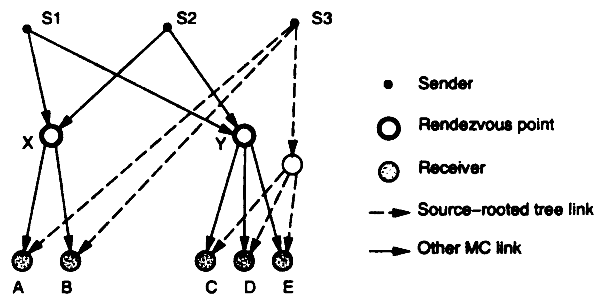


Figure 2.7: The result of topology transition for the sender S_3.

PIM's approach to supporting multiple MC topology types is elegant and efficient; we expect wide acceptance of the protocol in the Internet. However, its topology transition process, which builds SRTs, is data-driven and, hence, cannot be applied to connection-oriented networks, such as ATM networks, where routing must be established and maintained in a manner that is independent of traffic streams. The previous methods and current challenges of supporting group communication in such networks will be reviewed in the next section.

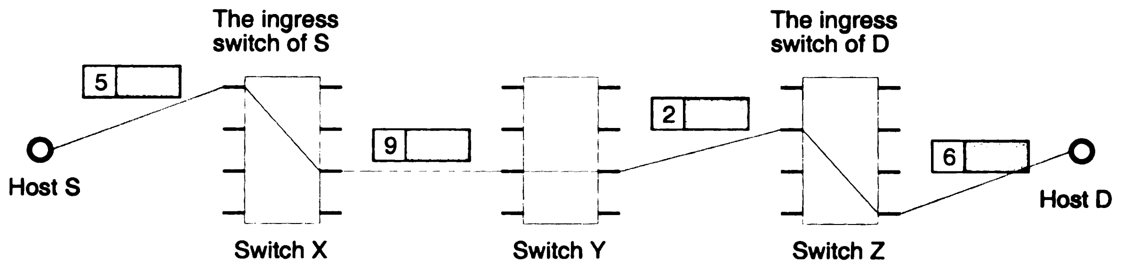
An important open issue regarding the PIM protocol is the selection of RPs and dissemination of their identities. According to the Internet multicast model described in [7], a host should be able to listen to a multicast address simply by informing its ingress router of the address. Since hosts are not obligated to provide RP identities, routers must obtain RP identities via an independent mechanism, which is not yet determined at the time of this writing [2]. As we will show in Chapters 6 and 7, modeling this RP management problem as a leader election problem within the network constitutes an important part of our research.

2.2.4 Multicast in ATM Networks

ATM networks are connection-oriented networks that relay small fixed-size cells in hardware. An ATM cell is 53-byte long, comprising 48 bytes of payload and 5 bytes of control information. Before transmission, a traffic source must set up a virtual circuit (VC) that defines a path between the source and a destination. All cells belonging to the VC will follow this path to reach the destination. Switching fabrics at ATM switches along the path use a *virtual circuit identifier* (VCI), contained in the control bytes of each cell, to determine the outgoing link for the cell.

These concepts are perhaps best explained using an example. Figure 2.8(a) depicts a VC between a source host S and a destination host D. In the example, we assume that every switch has four input ports, numbered 0 to 3, and four output ports, again numbered 0 to 3. Before transmission, the source host S issues a VC setup request to its ingress switch X; the conventions and procedures that a host follows to communicate with its ingress switch are termed the User-Network Interface (UNI) [30]. Included in the request message is an input-VCI field, which indicates the VCI value chosen by the requesting host to identify cells belonging to the VC. In the example, the source host S chooses the value 5. The ingress switch X determines an output port that leads to the next-step switch defined by a shortest S-to-D path (port 2, in this example, which leads to the switch Y), selects an unused output VCI value for the VC (9, in the example), replaces the value of the input-VCI field with the new value, and forwards the request to the next-step switch (namely, Y). The set of conventions and procedures that network switches use to communicate with each other is called the Private Network-to-Network Interface (PNNI) [13].

Continuing the example, the same task is repeated at switches Y and Z. Switch Y selects the output VCI 2, which becomes the input VCI for Z, and forwards the request to Z via port 2. Switch Z selects the output VCI 6, which becomes the VCI value that the destination host D uses to recognize cells pertaining to the VC. An S-to-D connection has been established. When traffic flows through the connection, an involved switching fabric uses a switching table to determine the forwarding of cells. The switching table at the port 0 of the switch X is shown in Figure 2.8(b). As we can see, the input VCI 5 is indexed into an entry that informs the switching fabric of X to forward cells with that VCI value to output port 2, and to tag those cells with the new VCI value 9. Further details of the VC setup procedure can be found in the UNI 3.1 [30] and PNNI 1.0 [13] standards, which have been produced by the ATM Forum, an international non-profit organization that comprises industrial and academic members.



(a) the use of VCI values along a path.

Input VCI	Output VCI	Output Port
⋮	⋮	⋮
5	9	2
⋮	⋮	⋮

(b) the switching table at port 0 of switch X.

Figure 2.8: VC operation in ATM networks.

The connection-oriented nature of ATM requires that the topology of an MC be determined and constructed before the presence of associated traffic streams. Further, the maintenance of the topology must be performed in a signaling-driven manner, that is, in response to network control messages, rather than the receipt of multicast data itself. For these reasons, many IP multicast solutions are not applicable to ATM networks. In this section, we discuss the ATM protocol used to establish one-to-many VCs, or multicast VCs, which are the only MC type presently supported by ATM standards. At the time of this writing, it is not clear which protocol(s) will be used in ATM to support other MC types. However, we will survey two proposals that have been discussed in the ATM Forum.

Multicast VCs

The concept of one-to-one VCs can be generalized to one-to-many VCs, or multicast VCs. This generalization requires an optional hardware feature, called *cell replication*, in order to forward multiple copies of an incoming cell via different output ports. This feature has been supported in many commercial ATM switches, for example, those provided by Fore Systems [40]. In UNI 3.1, a multicast VC has exactly one source party, called the root, and can be routed to one or more receiving parties, called leaves, following a tree topology. A multicast VC is set up by its root, which uses a procedure similar to the one-to-one VC setup procedure to connect to the first receiver. The result of this first step is a multicast VC with exactly one leaf node. Subsequently, the root can issue as many ADD-PARTY messages as necessary to attach additional leaves to the multicast VC. However, current ATM standards do not support group addresses, meaning that the source must learn the identities of receivers via a host-level protocol. In the most recent version of ATM UNI (namely, the UNI 4.0), receiver-initiated actions are supported so that receivers can join and leave a multicast VC without involving the source party. Again, receivers must learn via a host-level protocol the identities of the source party, or parties, in a multiparty communication application.

Proposals for Supporting Group Addressing in ATM

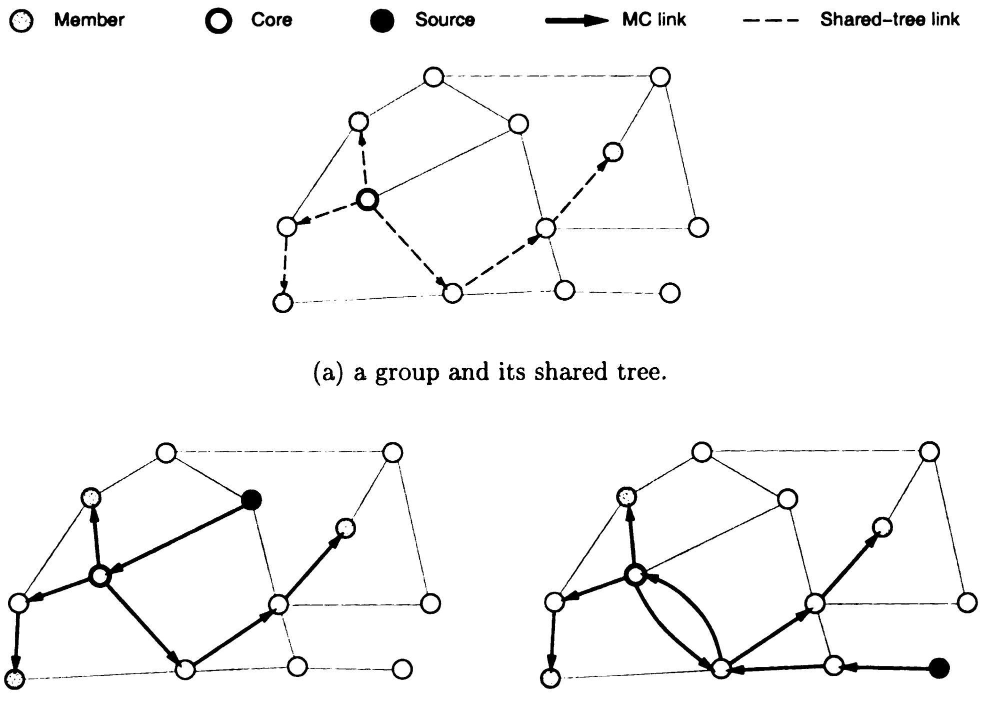
The lack of a group addressing mechanism in present ATM standards leaves the users/hosts to deal with the membership issue in group communication. The ATM Forum intends to add group addressing support in a future release of the PNNI standard [13]. Here, we review two proposals that have emerged within the ATM Forum.

1. A central-server approach for group membership management is promoted in [41]. In this proposal, a switch in an ATM network is configured as the group management center of the network, where the member lists of all active groups are maintained. Changes in membership must be sent to this switch in

order to update member lists. A host that wishes to construct a multicast VC to a group G contacts the management center to obtain the member list of G, and follows the UNI 3.1 standard to set up the multicast VC. This approach is designed for membership management, and facilitates the construction of multicast VCs, which are SRTs. Other MC topology types, such as receiver-only and symmetric shared trees, are still not supported. Further, the issue of single point of failure at the management center is considered "not critical," and is not addressed [41].

2. A variation of the CBT protocol for use in ATM networks, called the ACBT (ATM CBT) protocol, is described in [42]. This protocol is similar to the CBT protocol in that each group is assigned a core node, which is the root of a tree that reaches group members. This tree, however, is *not* an ATM multicast VC. Rather, the signaling modules of switches involved in the tree maintain the parent/child relations defined by the tree. In the ACBT protocol, a source party S can connect to all the members of a group via a single connection request, resulting in a multicast VC whose topology is the concatenation of a S-to-Core path and the shared tree rooted at the core. To illustrate, let us consider a three-member group shown in Figure 2.9(a), where the shared tree of the group is depicted by dashed lines. Figures 2.9(b) and (c) show the multicast VCs for two different sources. As shown in Figure 2.9(c), a link may be used by a multicast VC in two directions. This sometimes happens because the source must reach the core, the only contact point in the CBT and ACBT protocols, before the shared-tree can be used. We also emphasize that the two multicast VCs shown in the figure operate independently, despite the fact that they use identical sets of communication links (as defined by the shared tree) after a packet has reached the core node; the shared tree of a group exists in the form of signaling states, and is merely used to define the topology of multicast VCs destined to the group. Since multicast VCs destined to a group must be set up individually (although they share the same tree topology), it is difficult

to support some ATM features on a "per-group" basis. For example, given a group G, network resources must be reserved for each individual multicast VC destined to G, rather than for the group along.



(a) a group and its shared tree.

(b) an example of resultant multicast VCs.

(c) another example of resultant multicast VCs.

Figure 2.9: Operation of the ACBT protocol.

In summary, the ACBT protocol supports group addressing and multicast VCs, which are source-rooted but not necessarily shortest-path trees. Interestingly, the protocol, albeit a CBT variation, does not support shared-tree MCs. Another respect in which the ACBT protocol differs from the CBT protocol is the management of core. The ACBT protocol handles the selection of the core node when a group is created, rather than leaving the task to users/hosts, as in the case of the CBT protocol. When the first member of a group joins, the ACBT protocol randomly picks a switch as the core, and advertises this *core-group binding* via LSAs. This binding is recorded as part of the network image at every switch. Subsequent joining members follow a CBT-like procedure to connect themselves to the core, whose identity should now be available throughout the network. When different cores are suggested by several initial members that join the group at approximately the same time, the core candidate with the smallest ID wins.

2.2.5 Discussion

In summary, the designers of multicast protocols face the following challenges. First, multiparty communication applications demand a variety of MC topology types to meet different performance criteria. While multiple protocols could be used to achieve this goal, a single "generic" solution promises to avoid unnecessary overheads and redundancy. Second, it is desirable that host members of a group be aware only of the address of the group, and not the details of the underlying MC protocol. The fact that the group is associated with a core node, or a set of rendezvous points, should be hidden from users and hosts. As a result, any distinguished members needed in the protocol should be selected by the network, rather than by users or hosts. Third, when such distinguished members are required, the concern of a single point of failure arises. The network, rather than users and hosts, must handle such failures.

Presently, neither the IP multicast protocols nor the ATM solutions meet all these requirements. A main theme of this thesis is to show that these difficult issues in the Internet and in ATM networks can be appropriately addressed, when the network uses a specific type of routing, namely, link-state routing. Specifically, an LSR-based generic MC protocol will be presented in Chapter 5, and alternative approaches to modeling the RP/core management as a leader election problem in LSR-based networks will be discussed in Chapters 6 and 7.

2.3 Overview of Link State Routing

LSR was initially designed for use in the ARPANET [12]; fault tolerance issues associated with the original protocol are addressed in [43]. The ISO (International Standards Organization) version of LSR, the IS-IS (Intermediate System to Intermediate System) protocol [44], improves the efficiency of LSR when used in networks interconnected by broadcast-based LANs, such as Ethernet and token ring. These improvements have been incorporated in a new Internet routing protocol, called OSPF (Open Shortest Path First) [11]. Another recent application of LSR is the ATM PNNI standard [13], whose contributions include, among others, a method for hierarchically constructing large-scale, LSR-based networks, and an LSR-based group leader election protocol.

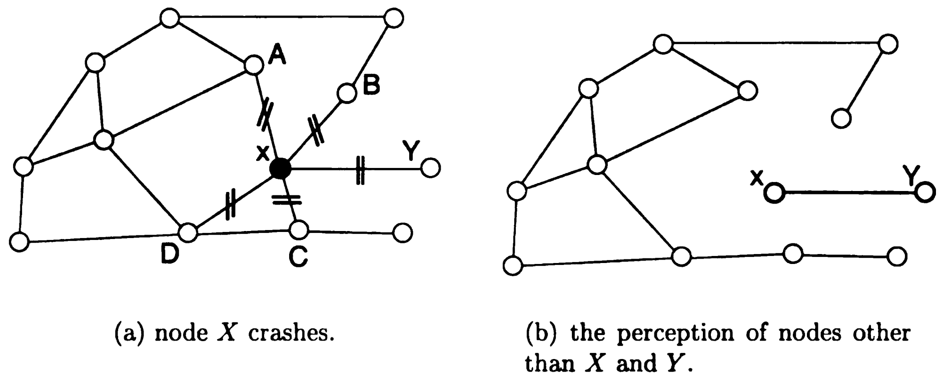
In this section, we provide background on LSR that will be needed later in the proposal. For purposes of discussion, the terms router, switch, and node will be used interchangeably.

2.3.1 Basic Operation

The essence of LSR is to maintain complete network images at all switches. For this purpose, every switch broadcasts throughout the network its local states, including *nodal* states and *link* states. Nodal states concern the working condition of a switch, for example, the workload at the switch. Link states describe communication links that are incident to the switch. Typically, link states include queueing delay, data loss rate, bandwidth, the capacity of associated buffers, monetary cost (for using the link), and so on. For historical reasons, control messages containing either state type are referred to as link-state advertisements (LSAs). After compiling an image of the network incrementally via received LSAs, a switch X routes traffic to a destination D according to a shortest X-to-D path computed locally. In general, the universal availability of complete network knowledge at every switch creates a robust infrastructure to support various network services, including group communication.

In order to update network images to reflect network status dynamics, every switch constantly monitors its local states and advertises changes in these states immediately. For example, when a link fails, the value of its working state is changed from ON to OFF, producing a *link-down* LSA from each of its endpoints. Similarly, *link-up* LSAs are flooded when the link later returns to an operational state. The working state of a link, which has only two values, is *discrete*; changes in such states are always advertised. For continuously valued states (such as queueing delay, which is a positive real number), a change in state is advertised only if the change exceeds a predetermined threshold.

The topology of a network is defined by the set of operational switches and communication links. Although it may be tempting to consider the working states of switches (as is the case for links), such states are not defined in LSR. That is to say, there are no "switch-up/switch-down" LSAs. This is because an LSR protocol cannot distinguish failed nodes from nodes that become unreachable due to failed links. To illustrate, let us consider the example in Figure 2.10(a), where the node X crashes. The five neighboring switches of X (A, B, C, D, and Y) detect the lack of responsiveness of the five links incident to X, and flood five respective link-down LSAs. In this example, switch A can learn of only four link-down events, because switch Y, which advertises the failure of the (X, Y) link, has been isolated by the failure of X. Figure 2.10(b) shows the network as perceived by switch A (and any other switches other than X and Y) at this moment of time.



(a) node X crashes.

(b) the perception of nodes other than X and Y.

Figure 2.10: Problem in correctly identifying node failure.

This observation suggests that an LSR protocol, which is not able to determine if a switch has failed or not, should instead be concerned with "reachability" to the switch. For example, once X and Y become unreachable, they cease to exist with respect to the operation of A. The concept of reachability is important not only to the handling of node failures, but also to the handling of much more disastrous circumstances, such as network partitioning. We will return to this issue in the next section.

A flooding protocol, used for the broadcast of network status information, is a highly robust protocol that guarantees that eventually all network nodes reachable from the source of an LSA will receive the LSA. The "conventional" flooding protocol works as follows. In order to send an LSA, the source switch sends the LSA to all its neighboring switches. For identification, LSAs typically contain the source address and a sequence number. When an LSA is received by another switch for the first time, it is forwarded on all incident links, except the one on which it arrived. Copies of LSAs that have already been seen by a switch are silently ignored. In this manner, every LSA is forwarded by every switch exactly once. An example of this flooding protocol is depicted in Figure 2.11; the flooding operation requires four steps to complete.

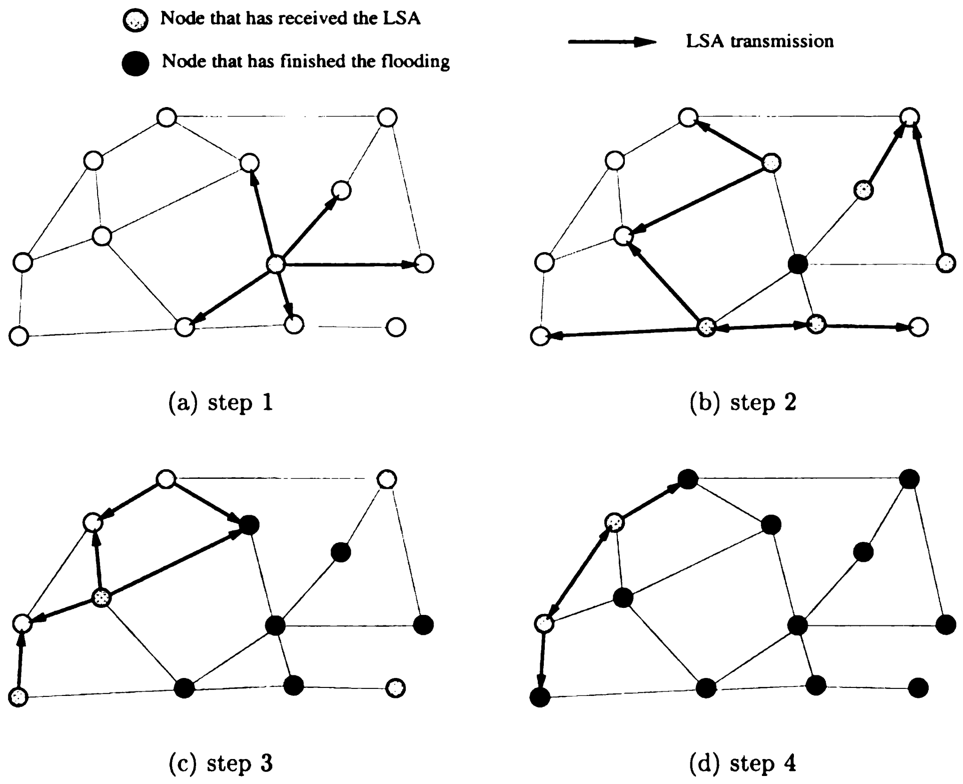

Node that has received the LSA
Node that has finished the flooding
LSA transmission
(a) step 1
(b) step 2
(c) step 3
(d) step 4

Figure 2.11: An example of the flooding operation.

The conventional flooding method has been adopted for use in both connectionless networks, such as the Internet, and connection-oriented networks, such ATM networks. In the case of ATM, the hardware-based multicast method, namely the use of multicast VC, has previously been considered unsuitable to the flooding/broadcast of LSAs, because it cannot guarantee the delivery of LSAs to all reachable nodes, as guaranteed by the conventional flooding method. Hence, the LSR operation in ATM proceeds in a less efficient, hop-by-hop manner. In Chapters 3 and 4, we will demonstrate how to take advantage of multicast VCs in flooding operations, while providing guaranteed delivery.

2.3.2 Fault Tolerance Issues

Networks are often expected to operate for long periods of time, in the presence of adverse conditions or even catastrophic scenarios. While many distributed applications ignore very rare adverse events, the networks themselves, and their underlying routing protocols, are expected to survive. Two types of such events, or faults, are of particular interest to LSR researchers: transmission errors not caught by the error detection mechanism (for example, CRC checksums) and the partitioning of the network.

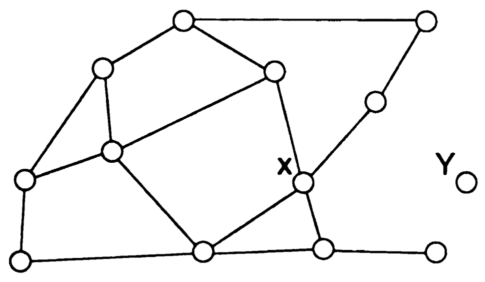
In LSR-based networks, the fault tolerance issue is closely related to the consensus problem. Recall that the consensus problem under LSR is to ensure the convergence of network images under the most adverse situations. Fault tolerance mechanisms in LSR either try to eliminate deterrents to achieving consensus or try to achieve consensus as soon as a consensus-prohibiting situation is cleared. A number of methods have been proposed to achieve this highly challenging goal [45]. Following is a summary of the widely-accepted OSPF solution [11]; a similar solution is adopted in the ATM PNNI standard [13].

- Switches not only advertise status changes immediately, but also broadcast their status periodically. This practice enables temporarily isolated segments of the network to exchange information with each other after re-unification (one segment learns of the existence of other segments in the next flooding cycle). Periodic flooding also controls the lifetimes of corrupted parts of network images that may occur due to undetected transmission errors, for the corrupted information will be overwritten in the next cycle of flooding.

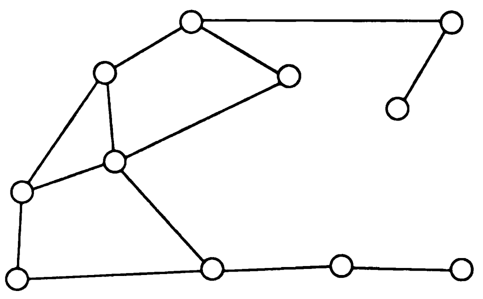
- An *aging* mechanism is used to identify obsolete information. Specifically, every entry in a network image has an associated aging timer, and the entry is discarded when its timer goes off. Nullified parts in a network image can later be filled by relevant LSAs with any sequence number value. The aging mechanism is needed to correct errors that the re-flooding mechanism along may take too long to correct. An example is undetected transmission errors in the sequence number field of LSAs. Let us consider an LSA with sequence number n that is incorrectly received as $n + k$ at some switch. Further assume that the source of the LSA re-floods every minute. If the value of k is 2^{28}, it would take more than 500 years for the source switch to catch up (that is, to use sequence numbers larger than $n+k$) and override the corrupted information. An aging mechanism solves this problem.

To further illustrate the use of these concepts, let us continue the example of Figure 2.10. Figure 2.12(a) depicts the local image at switch Y after the crash of X. We point out that the local image at switch Y (incorrectly) still contains links (X, A), (X, B), (X, C), and (X, D), because Y cannot receive corresponding link-down LSAs. An aging mechanism solves this problem. Using this mechanism, any node other than X and Y will remove the link (X, Y) and the nodes X and Y from its local network image, after not hearing periodic flooding from X and Y for a predetermined period of time. Put in another way, the $\{X, Y\}$ induced subgraph "ages out" in other parts of the network because it is no longer periodically reinforced by the two nodes. Figure 2.12(b) depicts the network image at any non-(X, Y) node after the aging mechanism takes effect. The network image at Y after aging consists of only one node, Y itself, since all the other nodes will age out at Y. This image is omitted in Figure 2.12.

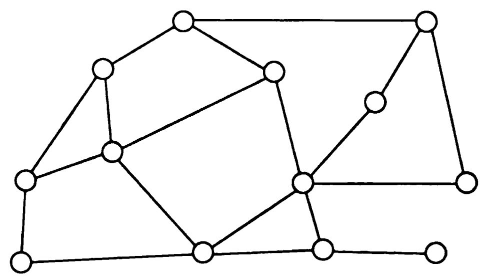
To finish the story with a happy ending, we assume that node X later becomes operational. After the revival of X, all switches learn of the existence of links incident



(a) network image at Y.



(b) network images at nodes other than X and Y, after aging.



(c) the consenting network image, after the revival of X.

Figure 2.12: The handling of network partitioning in LSR.

to X via link-up LSAs. Switches other than X and Y learn of the existence of these two nodes via the periodic status broadcasts from them. Similarly, the nodes X and Y become aware of the other parts of the network via periodic status broadcasts from other nodes. Eventually, all the switches will learn the network topology shown in Figure 2.12 (c), achieving consensus on the network images throughout the network.

The robustness of LSR is a major reason for its wide acceptance in many modern networks. However, the operation of LSR may raise concerns about scalability. First, the size of network images grows with the size of the network, which is consequently limited by the switch with the least memory space. Second, for a network with an average degree (the average number of incident links to nodes) d, every LSA will be received on average d times by every switch. Further, if the network has N switches that periodically flood their status for every T seconds, every switch needs to handle dN/T LSAs per second. When N is sufficiently large, the workload of LSA processing alone will exceed the computation capacity of switches, or the flooding of these LSAs may use up the bandwidth of the network.

Of course, there are ways to address the scalability issue. In the case of the Internet, LSR is 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 some other cases, such as the case of ATM, LSR is intended to support nation-wide, or even global, networks. In such cases, scalability can be achieved only by means of *hierarchical routing.*

2.3.3 Hierarchical LSR

Hierarchical routing reduces the burden on individual switches by hiding the complexity of the entire network. Different ways of supporting a routing hierarchy with LSR have been developed and deployed [11, 13]. In the Internet, the OSPF protocol defines a two-level LSR hierarchy such that a router sees only the subnetwork to which it belongs and the subnetwork's *boarder* routers, that is, routers that connect to the *backbone* subnetwork [11]. While intra-subnetwork traffic is routed as described in the previous section, cross-subnetwork traffic is routed in three stages: first through the home subnetwork to a boarder router, from there across the backbone network to reach a boarder node of the destination subnetwork, and finally through the destination subnetwork.

A more general method of hierarchical LSR is described in the ATM PNNI 1.0 standard [13], which allows for arbitrary hierarchy depth. In this method, a physical network is divided into several peer sub-networks, called *routing domains.* For example, the network shown in Figure 2.13 can be divided as shown in Figure 2.14. This division is performed manually by configuring every switch with a domain ID.

After division, each domain runs a separate instantiation of LSR, that is, switches within a domain exchange status information so that each of them maintains a "domain image." Continuing the previous example, the image of domain A.4 is depicted in Figure 2.15. As shown, a domain image contains not only intra-domain links, but also outgoing ones. An outgoing link, or inter-domain link, is advertised in the domains containing its endpoints. Hence, the link (A.4.1 A.2.3) in Figure 2.13 will be

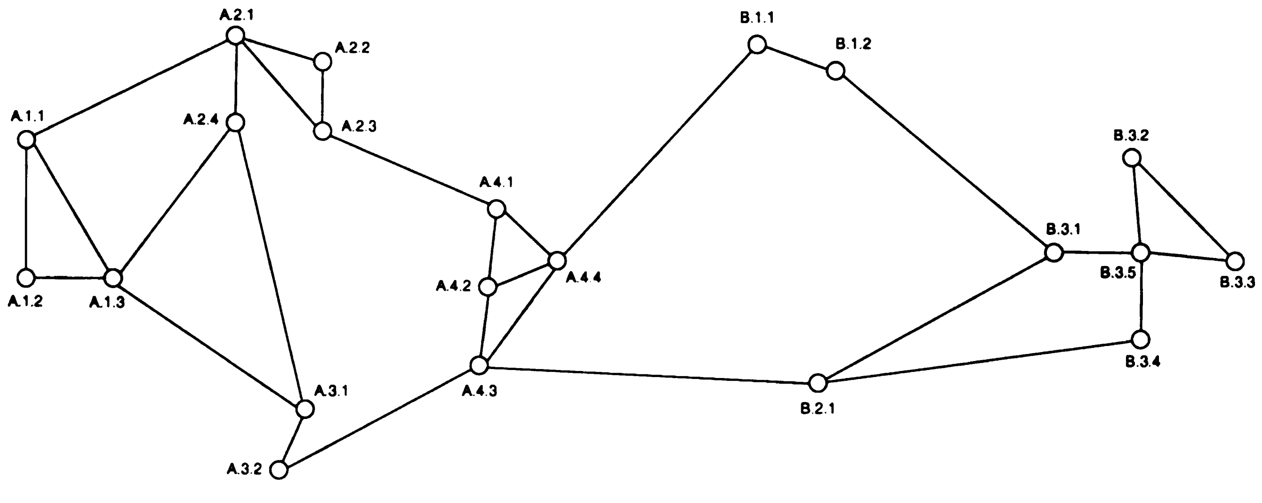


Figure 2.13: A network topology.

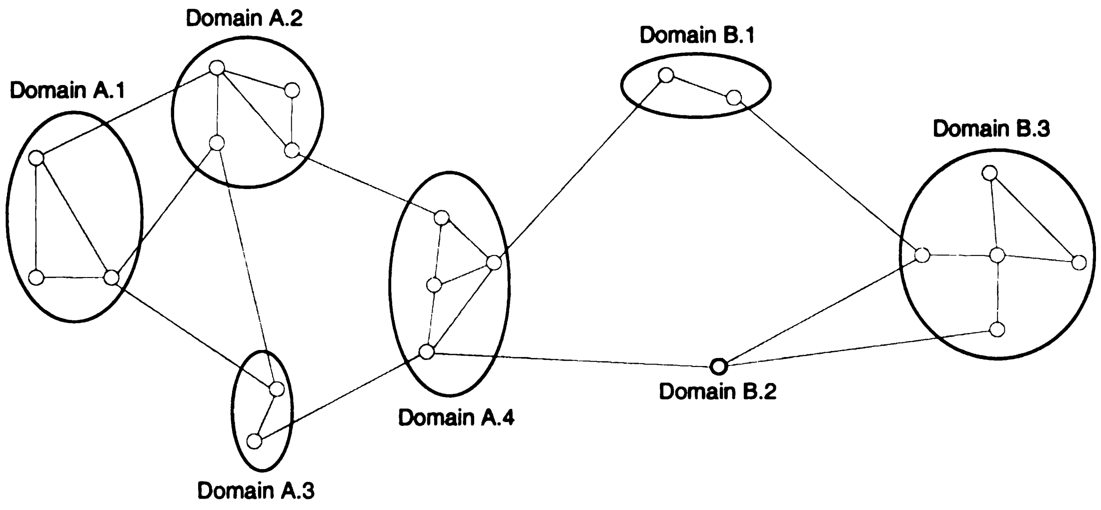


Figure 2.14: Breaking up the network into routing domains.

advertised in domain $A.4$ by switch $A.4.1$ and in domain $A.2$ by switch $A.2.3$. The presence of inter-domain links in the image of a domain enables the domain to see neighboring domains. For a domain to see all the other domains in the network, one must run a copy of inter-domain LSR.

To perform LSR among domains, a leader switch is elected within each domain. In ATM PNNI, the nodal states of a switch include two election-related states: *leader priority* and *preferred leader*. The former is manually configured by network managers to determine the rank of the switch. The latter is determined as follows: Every switch independently searches in its domain image for a reachable switch that has the highest leader priority, and calls the result of the search its preferred leader. As with other

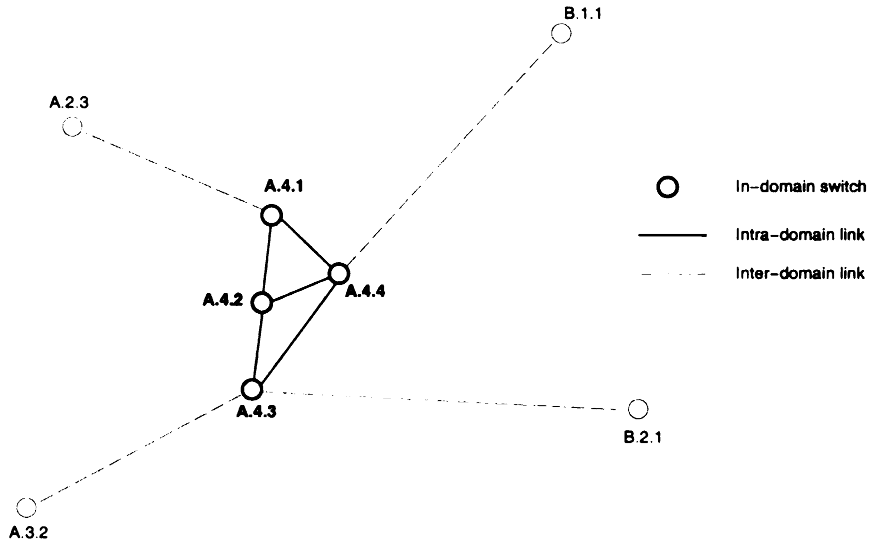


Figure 2.15: The image of the domain A.4.

LSR states, any change in the preferred leader state must be flooded immediately. If the preferred leader at a switch is the switch itself, this switch shall, after waiting for a period of time, inspect its local domain image for the preferred leaders of other switches. Only if unanimity is obtained will the candidate switch proclaim victory.

For illustration, consider a network where the administrator configures a default leader switch X with leader priority 3 and a backup leader Y with priority 2. The remaining switches are all configured with priority 1. We assume that initially switch X is the preferred leader of all other switches. Now consider what happens when the established leader X crashes. As described earlier, neighboring switches of X will advertise link-down LSAs for the incident links of X. Using these LSAs, every network switch finds the current leader unreachable, and searches through its local image for a switch with the next highest priority. In this case, the result would be Y with priority 2. Since every switch changes its value of the preferred-leader state to Y, every switch advertises this change immediately. These advertisements can be considered as "ballots," which the switch Y must collect before claiming itself the new leader.

Once elected, a leader learns the identities of neighboring leaders, namely the leader switches in neighboring domains, via the LSAs regarding inter-domain links.

(Preferred leaders of endpoints are included in such LSAs.) The leader then sets up a VC to connect to each neighboring leader. The inter-domain LSR is performed collectively by domain leaders as follows: each leader uses inter-leader VCs to flood to all the other leaders nodal states that present a simplified representation of its home domain and link states that describe its connectivity to neighboring domains. As such, each leader compiles a simplified view of the entire network. In this view, a node represents a routing domain and a link represents the adjacency of its endpoint domains. For the example of Figure 2.13, corresponding simplified network image is depicted in Figure 2.16.

In ATM PNNI, the division-and-simplification process just described can be applied recursively to build routing hierarchy of any depth. For example, when the network of Figure 2.13 is connected to an internet, the simplified network view shown in Figure 2.16 constitutes a domain in the internet, and a leader is elected among the domain leaders to represent the entire network in the next routing level.

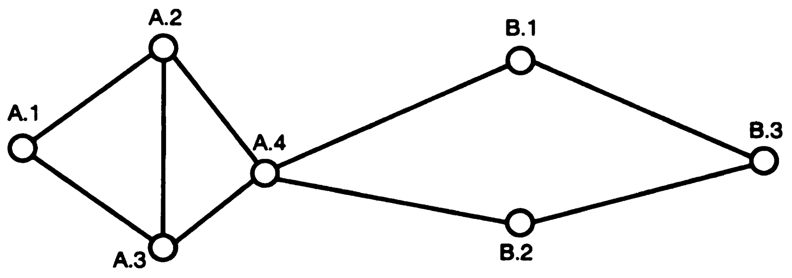


Figure 2.16: The simplified/high-level network image.

2.4 Discussion

The main theme of this thesis is to demonstrate and exploit the mutually beneficial relationship between group communication and LSR. Three facets of group communication will be examined for being supported by LSR: use of multicast VCs in LSR flooding, MC construction and maintenance, and leadership consensus. Let us now briefly introduce each of these problems, given the background information that has been presented in this chapter.

First, LSR itself can benefit from group communication techniques, because many aspects of LSR operation exhibit characteristics of group communication. In LSR, switches in a routing domain form a communication group: they broadcast to the group, receive broadcast messages (that is, LSAs) from the group, maintain member lists of the group (which are implicitly included in local domain images), and elect a leader to represent the group in the next routing level. Moreover, such group communication characteristics in LSR are even more obvious in hierarchical LSR networks: At higher routing levels, the LSR tasks of flooding, membership management, and leader election are performed collectively by domain leaders. Since leaders are not necessarily physically adjacent with each other, a flooding operation among leaders forms a true multicast operation in the entire network. In this thesis, we identify an important aspect of LSR that can benefit from group communication: the flooding operation. We note that, while present ATM standards use hardware switching and cell replication to speed up host-level multicast, flooding operations still proceed in a store-and-forward manner as described earlier. Our first main contribution is to show that flooding operations can make use of the hardware capability of ATM switching fabrics to improve performance, while in the meantime guaranteeing delivery to all nodes reachable from an originating node, as in the case of the conventional flooding protocol. In Chapters 3 and 4, we describe a family of *switch-aided flooding* (SAF) protocols that work in this manner.

Second, the construction of MCs can benefit from the complete network information made available by LSR. We have discussed one multicast protocol, the MOSPF protocol, that takes advantages of LSR; it uses LSR to disseminate membership information so that every router has a member list for every active MC. However, the MOSPF protocol is restrictive in supporting different MC topology types, and incurs computational redundancy. As we noticed in previous sections, multiparty communication applications need different MC topology types. Further, the rising importance of QoS service is leading to new, sophisticated MC topology computation algorithms, many of which are not supported by existing MC/multicast protocols. This thesis will show that the availability of complete network and MC membership information at switches/routers in LSR-based networks makes it possible to design a "chassis" for MC protocols to accommodate existing and future MC topology computation algorithms. The resultant *generic MC* (GMC) protocol will be presented in Chapter 5.

Third, we consider the problem of leader election. Although leader election is not directly required by all group communication applications, some prominent multicast protocols, such as the CBT and PIM, assign a network node as the multicast traffic transit center, or the core node, for the group. Arguably, the core node of a group must be selected by the network; if the identity of the core is provided by host members, then the host-network interface for multicast depends on the choice of multicast protocol within the network (some multicast protocols require core identities from the interface, while others do not). Further, the introduction of a traffic transit center raises the concern of single point of failure.

The problem of assigning of core nodes to groups can be modeled as a leader election problem (the leader of a group undertakes the responsibility of the core node). The fault tolerance of LSR enables the design of robust election protocols, such as the ATM leader election protocol, that handle not only leader failures but also disastrous scenarios, for example, network partitioning. However, the overhead of the current ATM leader election protocol (every group member uses flooding to report its preferred leader) may be prohibitively expensive if used to support multicast groups because a large number of such groups may exist simultaneously in a network. The design of efficient LSR-based support for the election problem constitutes the third part of this research. Our NLE protocol, presented in Chapter 6, accommodates a membership management mechanism that achieves the following consensus property: a set of mutually reachable group members reach consensus on a leader, which maintains a member list containing exactly those members. The LCM protocol, presented in Chapter 7, uses the NLE protocol to elect a leader switch as the centralized core management server, which manages the core nodes for all active groups within the network.

Finally, we come full circle. 'By combining two group communication techniques developed earlier, namely the election of a leader and the construction of multipoint connections, we develop a totally different approach to LSR. The resulting *Tree-based LSR* (T-LSR) protocol is lightweight, imposing only a small fraction of the overhead of previous LSR methods, and robust, guaranteeing to survive not only network component failures and partitioning scenarios, but also undetected communication transmission errors. As we discussed earlier, properly handling the latter type of faults is a vital requirement for an LSR protocol. Unlike the ATM-oriented SAF protocols, the T-LSR protocol is designed for use in general-purpose, LSR-based networking environments and requires no special hardware support.

At the first glance, the advocation of group-communication-supported LSR operations and LSR-based group communication introduces a "chicken and egg" dilemma — which one should exist first so as to support the other? Our results show that, with careful design, the circular dependence can be avoided. The SAF and T-LSR protocols demonstrate how a multiparty communication channel can be constructed and used to improve the performance of flooding operations, which advertise routing information (namely, LSAs) necessary for the construction and maintenance of the channel. On the other hand, the GMC protocol can take advantage of LSR performance improvements by T-LSR and SAF methods to enable the use of any topology computation algorithm and hence provide support for any MC topology type. Moreover, the NLE protocol, which itself is LSR-based, finds applications in both the internal operations of LSR (such as hierarchical routing) and the support of multiparty communication applications (for instance, the management of multicast cores used by such applications). Such results demonstrate the mutually beneficial relationship between LSR and group communication.

Chapter 3

Switch-Aided Flooding

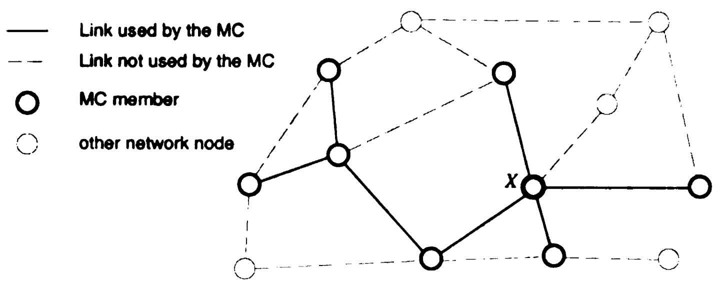
In this chapter, we demonstrate an example to support the claim that some aspects of LSR operation can benefit from group communication. Specifically, we propose a flooding method, called Switch-Aided Flooding (SAF), for use in ATM networks. SAF-based protocols take advantage of hardware-supported cell relay and cell duplication, characteristic of such networks, in order to reduce the time needed to disseminate changes in network topology and resource availability. SAF protocols use a spanning multipoint connection (SMC), which is a hardware-switched network spanning tree, but revert to conventional link-by-link flooding when the spanning MC is unavailable or under construction. Two flooding protocols based on this methodology, as well as an accompanying protocol to construct and maintain the SMC, are described in this chapter; a third SAF protocol is described in Chapter 4. The results of a simulation study reveal that the proposed flooding protocols deliver network updates several times faster than conventional approaches. Further, the bandwidth consumed by a flooding operation is also significantly reduced.

3.1 Motivation

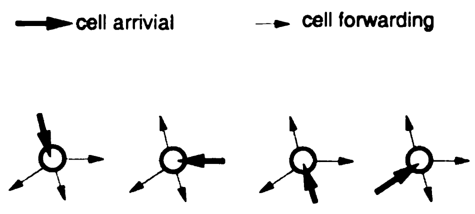
As described in Chapter 2, ATM is a connection-oriented communication technology that relays small fixed-size cells in hardware. Many ATM switching fabrics support hardware cell duplication, whereby an incoming cell can be forwarded via multiple output ports. Although current ATM standards use this feature to support only multicast (or one-to-many) VCs, such switch functionality enables the construction of a more generic form of group communication channels, namely, many-to-many VCs, or *multipoint connections* (MCs). An example of an MC is depicted in Figure 3.1(a), where a set of eight switches is interconnected with a tree topology. The responsibility of each member switch is to forward cells arriving on one link of the tree to all the other tree links that are incident to that switch. As illustrated in Figure 3.1(b), cells arriving on any of the four links incident to the switch x are forwarded on the remaining three incident links. Hardware-supported MCs facilitate multiparty communication applications, such as multimedia teleconferencing, distributed virtual reality, tele-gaming, and computer-supported cooperative work. MCs used in such applications typically involve only a small subset of the network switches. A special type of MC is the *spanning* MC (SMC), which includes as its members all switches in a network. A spanning MC of the network used in Figure 3.1(a) is depicted in Figure 3.1(c). Since every message transmitted on an SMC is received by all switches, the SMC can be considered as a virtual broadcast medium of the network.

Although hardware switching and cell duplication may greatly improve the communication performance observed by end hosts and their applications, the signaling activities within ATM networks, as defined in UNI 3.1 [30] and PNNI [13] standards, proceed largely in a connectionless manner. Since signaling must take place prior to the existence of corresponding VCs [30], VC-setup request messages are forwarded and processed in a hop-by-hop manner. Switches along the route of the VC under construction invoke signaling modules to perform functions related to the requested VC, such as routing and call admission control. In addition, the ATM PNNI standard specifies the use of the flooding protocol described in Chapter 2, which was originally designed for the ARPANET, a connectionless point-to-point network. Not surprisingly, the protocol proceeds in a hop-by-hop manner, and does not take advantage of the hardware capabilities of ATM switching fabrics.

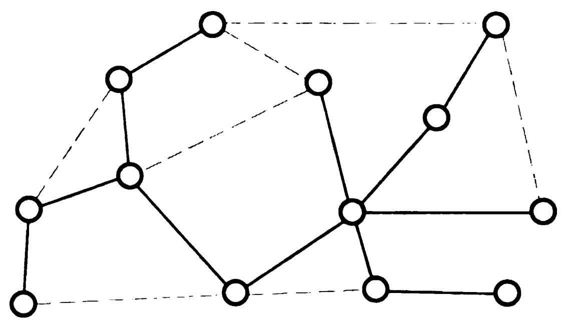
We model the ATM flooding operation as a group communication problem, where an LSA is considered as a multicast message delivered to a group comprising all



(a) an 8-node MC



(b) cell forwarding at switch x



(c) a spanning MC of the network

Figure 3.1: Examples of multipoint connections.

switches in the network. The proposed SAF method uses a common group communication topology, the tree topology, to facilitate the dissemination of LSAs. Specifically, the SAF method constructs a spanning MC, which is used as a "broadcast medium" for distributing LSAs. The use of an SMC improves the performance of flooding operations by taking advantage of both hardware cell relaying and cell replication. However, such an approach must address the challenge of retaining the robustness of the conventional flooding method, that is, an LSA must reach all switches reachable from the source of the LSA.

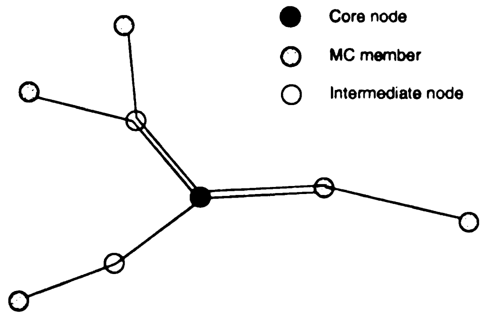
The main contribution of this chapter is to develop and evaluate two SAF-based flooding protocols, called *Basic* SAF and *bandwidth-efficient* (BE) SAF protocols, that satisfy these criteria. In addition, an efficient protocol for the construction and maintenance of spanning MCs is presented. The results of a simulation study reveal that these two SAF-based flooding protocols can distribute messages to network switches several times faster than the conventional flooding algorithm. In the next chapter, we will develop a even more efficient SAF protocol by using a second group communication topology, the ring topology, to implement reliability. A robust and efficient flooding protocol can lead to better routing decisions by reducing reaction time to faulty network components and congested areas. This in turn reduces the probability of call blocking. Furthermore, general-purpose, LSR-based MC protocols, such as the MOSPF protocol [6] and the GMC protocol (discussed in Chapter 5), must disseminate group membership and/or MC topology advertisements, and therefore can also benefit from efficient flooding protocols.

The remainder of this chapter is organized as follows. A protocol that constructs and maintains a network-wide spanning MC is presented in Section 3.2. In Section 3.3, two SAF protocols are presented. The Basic SAF protocol extends the conventional flooding algorithm to incorporate the use of an SMC. The BE SAF protocol further addresses the issue of bandwidth consumption in flooding operations. The performance of these two protocols is investigated through a simulation study, the results of which are presented in Section 3.4. A summarization of this work is presented in Section 3.5.

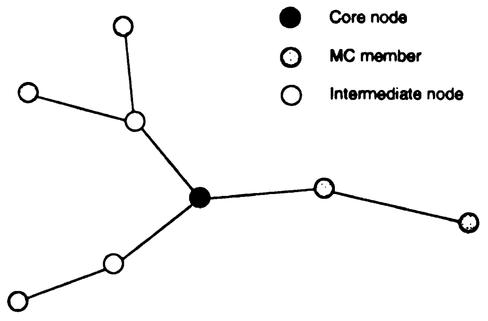
3.2 The Spanning MC Protocol

The SMC protocol constructs and maintains an SMC for use in the SAF protocols. The protocol is a variation of the CBT protocol [3, 4], a general MC protocol in which the topology of the MC is the union of shortest paths from the members to a specific node, called the *core* (see Figure 3.2). The SMC protocol differs from the CBT protocol in the way that the core node of the MC is determined. In the CBT protocol, the core node is static and is determined by an "outside" mechanism (for example, by network management procedures). In the SMC protocol, the core node is dynamic for reasons of robustness, since the SMC protocol must survive extensive network changes, including failure of the core node itself.

In the SMC protocol, the core node selection problem is modeled as a leader election problem under LSR. In this approach, every switch x uses the same core node



(a) the member-to-core shortest paths.



(b) the resultant MC topology.

Figure 3.2: An example MC built by the CBT protocol.

selection algorithm to independently identify a new core of the SMC. The choice of switch x will be referred to as c_x, and the computation will be denoted by $C(G, x)$, where G is the network image at x. For now, we use the function $C(G, x)$ that simply sets c_x to the preferred leader at switch x, that is, we use the domain leader switch elected by the ATM PNNI as the core node of the SMC. The generalization that allows the use of any core selection algorithm $C(G, x)$ can be achieved by using our Network-level Leader Election (NLE) protocol, which is discussed in Chapter 6. Discussion and evaluation of a variety of core selection heuristics can be found in [46, 47].

After selecting the core node locally, each switch tries to establish a connection to its choice of core node. For a switch x to reach its core selection c_x, the switch sends a **reach_core** request one hop towards the core, according to an x-to-c_x shortest path computed locally. The receiving switch grants the request after it has successfully reached the core itself. Using the network shown in Figure 3.1 as an example, the process of SMC construction is illustrated in Figure 3.3. Let us assume that all nodes initially select, as the core, the darkened node in Figure 3.3(a); this figure also shows the direction of sending **reach_core** requests. The core node immediately grants the **reach_core** requests from its neighboring switches, which subsequently approve **reach_core** requests from downstream switches. In this way, SMC links are granted and established in a "radiating" manner; see Figures 3.3(b) to 3.3(f).

Under the SMC protocol, each switch x in the network G executes a set of con-

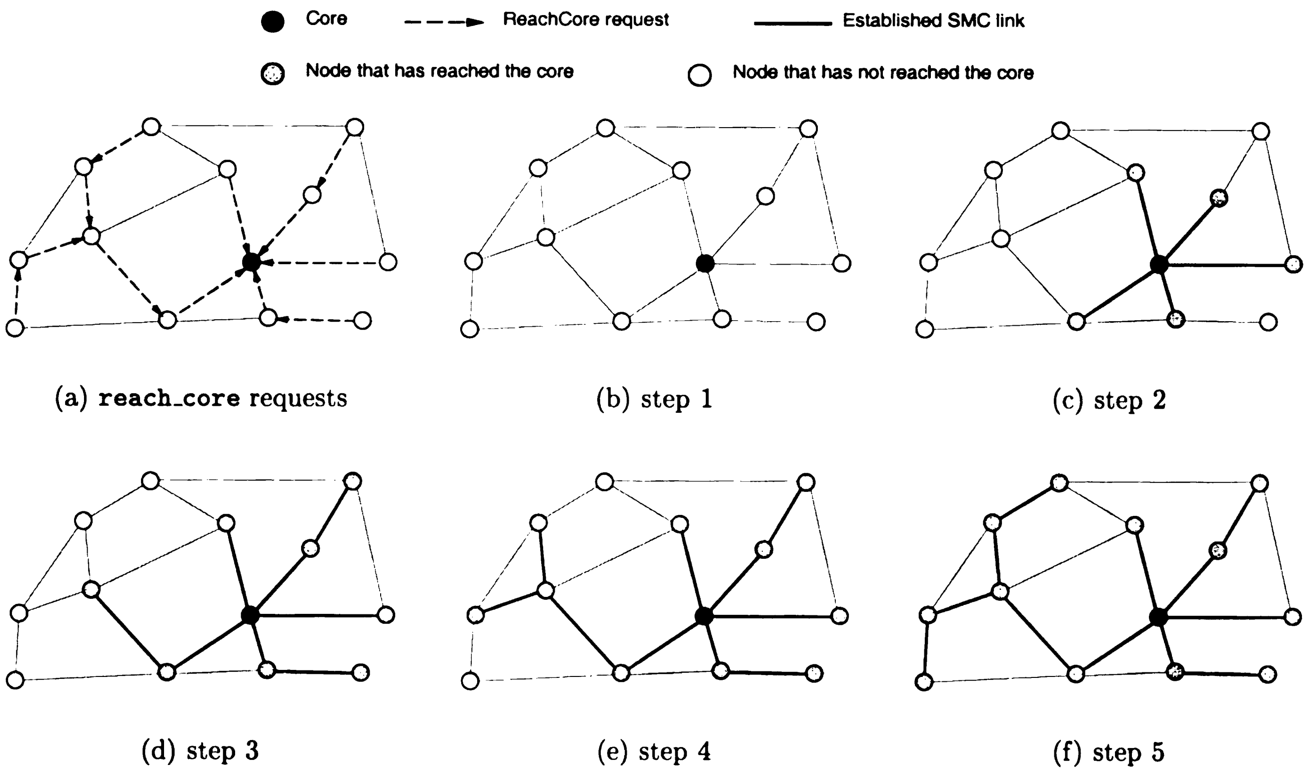


Figure 3.3: An example of the SMC protocol.

stituent protocol modules and maintains the following data structures: a local network image G_x, a core selection c_x, and an x-to-c_x path P_x. (In the following, we may omit the subscripts if they are clear from context.) Whenever a data structure must be accessed by concurrent protocol modules, access to the data is assumed to be *atomic*, in order to avoid race conditions among protocol entities. Critical regions and semaphores are well-known techniques to achieve atomic access.

SMC protocol operation is triggered by the receipt of an event LSA (link-down, link-up, and so on). Periodic LSAs are ignored by the SMC protocol so that the protocol, and hence reconstruction/reorganization of the SMC, will not occur unnecessarily. As shown in Figure 3.4, upon receiving an event LSA, the SMC protocol at switch x updates the local image G_x of the network. The protocol then decides whether it has to re-connect to the core node because 1) its core selection changes, or 2) the LSA ℓ reports a failed link that is used in P_x. When it is necessary to re-connect to the core, the switch x tears down the present MC link that leads to the core node and initiates an attempt to reach the core node by signaling another protocol entity, the **ReachCore** module. We emphasize that the inclusion of maintaining network image G_x in SMC algorithms is for the purpose of self-contained discussion; in real-world contexts, G_x is most likely maintained by the underlying LSR protocol.

```
Algorithm: Process_Event_LSA.
Input: switch ID $x$, received LSA $\ell$.

Update $G$ according to $\ell$.
IF ($c_x \neq C(G,x)$) or (LinkDown($\ell$)=TRUE and Link($\ell$) in $P_x$)
    Let $y$ be the next hop to $c_x$ in $P_x$.
    Disconnect the tree link $(x,y)$.
    $c_x = C(G,x)$.
    Wake up the ReachCore module if it is sleeping.
ENDIF
```

Figure 3.4: The handling of event LSAs.

The **ReachCore** module at switch x is started after the initialization of x, and loops indefinitely. This module is responsible for setting up an SMC link that will lead to c_x. For this purpose, the module sends a `reach_core(`c_x`)` message one step towards the core, and will continue doing so until a positive reply is received from the appropriate neighbor, indicating that the request has been granted and the desired link established. We note that the value of c_x may change during this period, because the **Process_Event_LSA** module may update the value upon receiving new event LSAs. After obtaining a positive reply, the **ReachCore** module records the new to-core path, P_x, and suspends itself.

The routine that processes a `reach_core` message is shown in Figure 3.6. The receipt of such a request from switch y by switch x suggests that the switch x is the first intermediate node on the path from y to c_y. The switch x grants the request if 1) it agrees with y upon the choice of core node, and 2) it has itself reached the core (this can be determined by whether the **ReachCore** module at x is suspended). When the request is granted, the switch x establishes the (x,y) MC link and returns a positive reply to y, which includes the y-to-c_y path used by the SMC. (The establishment of an MC link involves the setup/modification of hardware switching table entries to implement the type of cell forwarding depicted in Figure 3.1(b).) Otherwise, a

```
Algorithm: ReachCore.
Input: switch ID $x$.

LOOP forever
    IF ($C_x \neq x$)
        LOOP
            /* note: $C_x$ may have been changed by Process_Event_LSA */
            Let $y$ be the next stop to reach $C_x$.
            Send a reach_core($c_x$) message to $y$.
            Wait for a reply.
        UNTIL (a reply reached_via($P$) is received).
        $P_x = P$.
    ENDIF
    Sleep.
ENDLOOP
```

Figure 3.5: The ReachCore module

negative reply is returned.

```
Algorithm: Process_Reach_Core.
Input: switch ID $x$ and a reach_core($c$) request from switch $y$.

if ($c = c_x$) and (I've reached the core node $c_x$)
  Setup the $(x, y)$ MC link.
  Return a positive reply, reached_via($P_x + (x, y)$), to $y$.
ELSE
  Return a negative reply to $y$.
ENDIF
```

Figure 3.6: The processing of the `reach_core` request message.

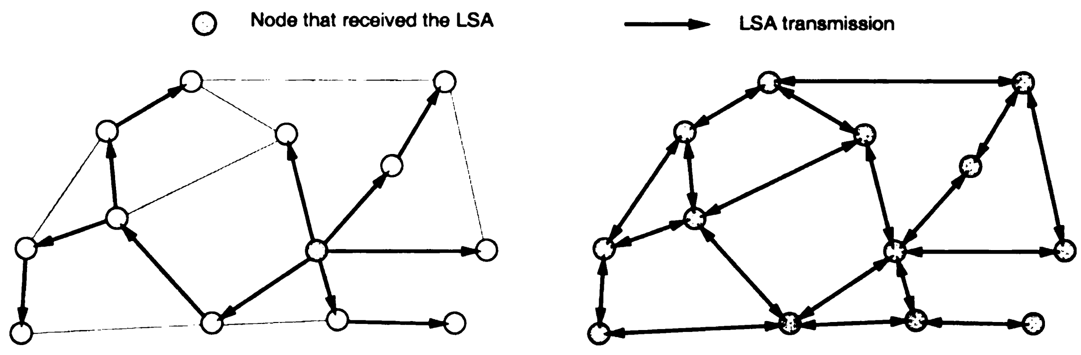
Cell Demulplexing. Because a spanning MC is effectively a broadcast medium that allows interleaving of messages, every switch in the network can broadcast messages to, and receive messages from, all other switches. However, cells belonging to simultaneous broadcast messages can be interleaved with one another at intermediate switches. Receiving switches must be able to demultiplex these messages according to their sources. Various methods can be used to solve this problem. For example, part of the cell payload can be used to label the sources of cells. Alternatively, spanning MCs can be switched by the *virtual path identifier* (VPI). In ATM networks, every cell is tagged with a pair of identifiers, VPI and VCI. When the VPI of a VC is used in switching, the VCI of the cells belonging to the VC is ignored (but remains intact during transmission). In this approach, the SMC used by the SAF protocol must be constructed in such a way that the VPI is in effect throughout the MC, and as such, the VCI field can be used to identify the source switch of cells. We emphasize that the SMC protocol, and the SAF protocols as well, work with any demultiplexing scheme.

3.3 The SAF Protocols

An SAF protocol is an extension of the conventional flooding protocol. In addition to a set of point-to-point links in a network, SAF protocols presume the existence of an SMC to which all the switches in the network have access. In this section, we present two protocols designed in this manner; they differ in the implementations of reliability.

3.3.1 Basic SAF Protocol

This protocol works as follows. The source of an LSA first broadcasts the LSA on the SMC and subsequently sends the LSA via all its incident links. If a switch receives the LSA for the first time via the SMC, then it forwards the LSA on all its incident links. On the other hand, if the switch receives the LSA via a point-to-point link, then it forwards the LSA on all incident links except the one on which the LSA arrived. As in the case of conventional flooding, switches silently drop LSAs that have been seen previously. To illustrate, the flooding example of Figure 2.11 is repeated in Figure 3.7, but this time using the Basic SAF protocol. As we can see in the figure, the operation now requires only two communication steps. In the first step, the source switch broadcasts the LSA, which is switched and duplicated in hardware on the SMC. In this manner, the constituent cells are pipelined throughout the network. Provided that other switches receive the LSA in the first step, they exchange this LSA via point-to-point links in the second step; since every node has already seen the LSA via SMC broadcast, all the point-to-point copies are dropped.

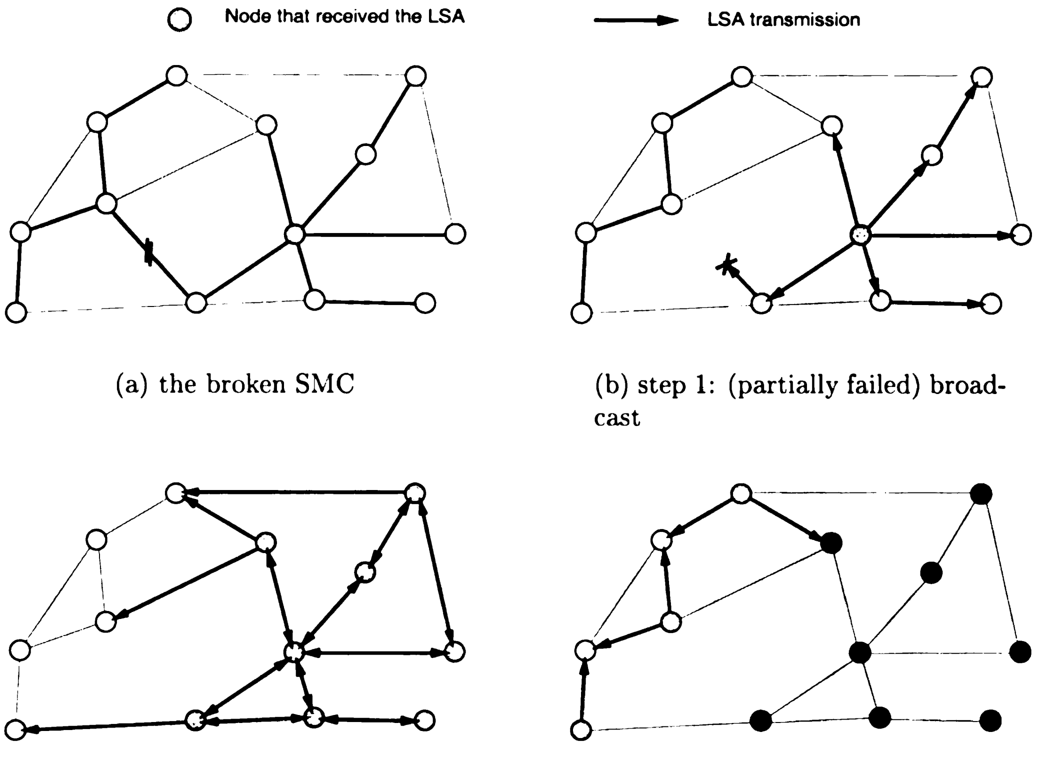


(a) step 1: hardware switched broadcast (b) step 2: point-to-point forwarding

Figure 3.7: An example of the Basic SAF protocol.

The Basic SAF protocol uses the SMC as a shortcut for LSA dissemination, but does not rely on this shortcut. In normal cases, such as the one in Figure 3.7, switches receive LSAs immediately via the SMC. However, in situations where one or more links used in the SMC is malfunctioning, or the SMC itself is under construction, or cell losses occur on the SMC, then the link-by-link forwarding will guarantee that the LSA reaches all nodes. Shown in Figure 3.8 is an example of how the Basic SAF protocol operates when the SMC is faulty. In this example, a link that is used in the SMC fails during a flooding operation and the broadcast of the LSA cannot reach all switches (Figures 3.8(b) and 3.8(b)). As shown in Figures 3.8(c) and 3.8(d), the remaining switches are reached via link-by-link forwarding. In extreme cases where the SMC does not exist at all (for example, when the network is re-initialized), the Basic SAF protocol degenerates to the conventional flooding protocol.

The Basic SAF achieves its efficiency at the price of additional bandwidth consumption. Here we compare the bandwidth used by the conventional flooding protocol against that of the Basic SAF protocol. In the conventional flooding protocol, the source of an LSA sends the LSA on all its incident links, and other nodes forward the LSA on all but one of the incident links. Consider a network $G = (V, E)$, where V is the set of switches and E the set of point-to-point links. The number of links



(a) the broken SMC

(b) step 1: (partially failed) broadcast

(c) step 2: link-by-link forwarding

(d) step 3: link-by-link forwarding

Figure 3.8: The Basic SAF protocol with a broken SMC.

traversed by conventional flooding is

$$B_c = 1 + \sum_{v \in V} (Deg(v) - 1) = 1 + Deg(G) - N,$$

where $N = |V|$ and $Deg(G)$ is the sum of node degrees in G. On the other hand, the Basic SAF protocol would require

$$B_{\text{basic}} = (N - 1) + \sum_{v \in V} Deg(v)$$

links, where the first term $(N - 1)$ is the number of links used by the broadcast on the SMC, and the second represents forwarding of the LSA on point-to-point links.

To further clarify the relationship between B_c and B_{basic}, let $AvgD$ denote the average node degree of G. The bandwidth consumptions of the two flooding protocols can be rewritten as

$$B_c = AvgD \times N - N + 1 \approx (AvgD \times N) - N,$$

and

$$B_{\text{basic}} = AvgD \times N + N - 1 \approx (AvgD \times N) + N.$$

Therefore, the B_{basic} to B_c ratio can be approximated by

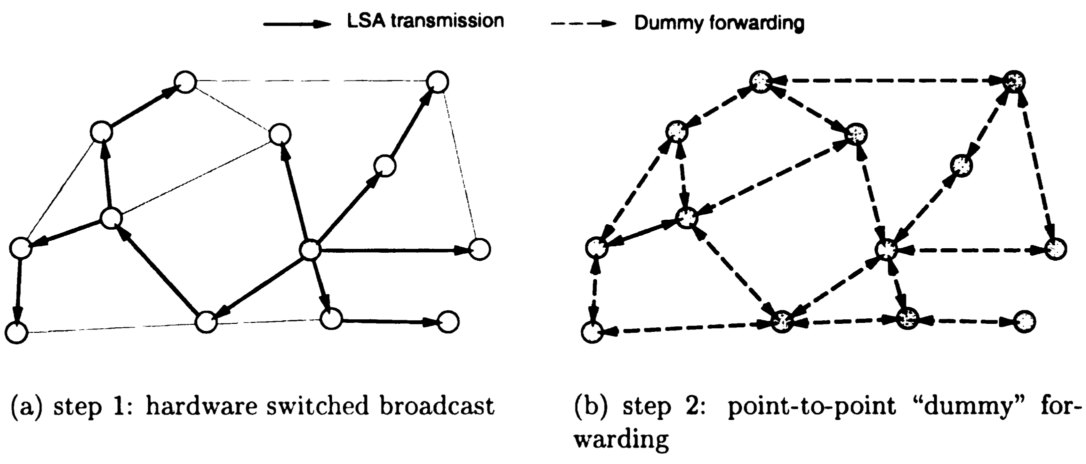
$$\frac{B_{basic}}{B_c} \approx \frac{AvgD + 1}{AvgD - 1}.$$

If the network has a small average node degree, then the Basic SAF protocol may consume significantly more bandwidth than does the conventional flooding protocol. However, due to the simplicity of this protocol and its advantage in flooding time, the Basic SAF protocol may be attractive under a variety of conditions (see Section 3.4).

3.3.2 Bandwidth-Efficient SAF Protocol

The Basic SAF protocol can be modified to reduce bandwidth consumption by introducing the concept of *dummy forwarding*. In this approach, switches receiving an LSA via the SMC forward a "dummy" of the LSA, containing only the source address and sequence number, to neighboring switches. Switches that have finished the task of dummy forwarding also expect to see responses (either the real LSA or its dummy) from all neighboring switches. After waiting for a predetermined period of time, such switches forward the real LSA to neighboring switches that fail to respond. Switches receiving the LSA via point-to-point links forward the real LSA on all incident links except that of arrival, and expect nothing from neighbors.

Again we use examples to illustrate. The operation of the BE SAF protocol with a fully operational SMC is depicted in Figure 3.9. Similar to the Basic SAF protocol, the BE SAF protocol in this setting requires only two communication steps. In the first step, the source switch broadcasts the LSA, which is switched and duplicated in hardware on the SMC. In the second step, however, switches exchange with neighboring switches dummies of the LSA, rather than the real one.



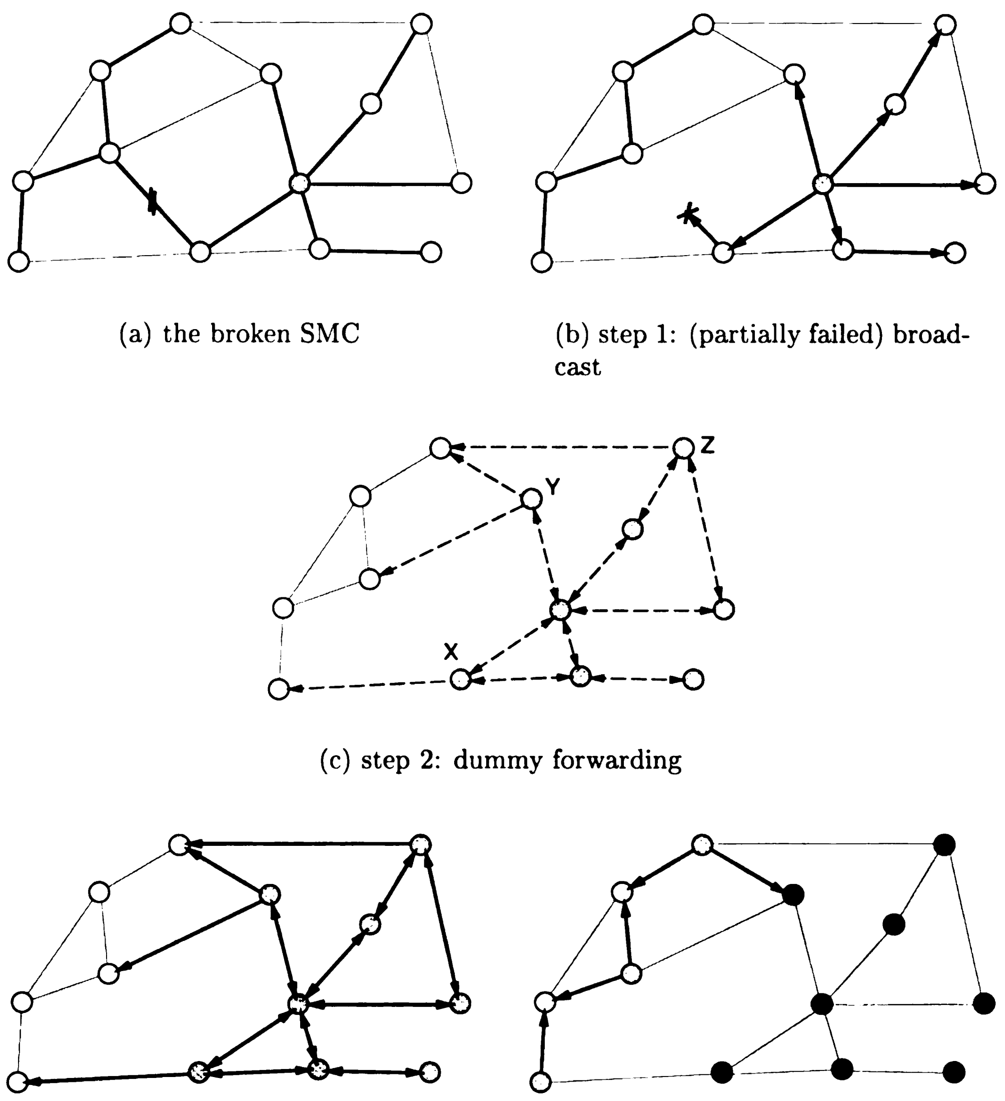
(a) step 1: hardware switched broadcast (b) step 2: point-to-point "dummy" forwarding

Figure 3.9: An example of the BE SAF protocol.

Figure 3.10 illustrates the operation of the BE SAF protocol with a broken SMC. After the BE SAF protocol uses the SMC to reach as many nodes as possible (Figure 3.10(b)), switches that received the LSA via the SMC forward dummies, and in the meantime expect their neighboring switches to do the same. In this example, three nodes, namely X, Y, and Z, do not see all the expected dummies from neighboring switches; see the unidirectional dummy forwardings in Figure 3.10(c). The three nodes, after a predetermined timeout period, start forwarding the real LSA to their "silent" neighbors, as depicted in Figure 3.10(d). Switches that receive the LSA via link-by-link forwarding further forward the LSA to all incident links except the incoming one, as shown in Figure 3.10(e).

In the discussion of BE SAF algorithms, we denote by K_x the number of switches that are neighbors of switch x. Let us assume that the neighbors of the switch x can be reached via ports numbered 1 to K_x, and that the SMC is attached to port 0.[1] A switch maintains three data structures: Seq[i], the sequence number of the current LSA from switch i, $1 \leq i \leq N$; Received[i], a boolean flag indicating if the Seq[i]-th LSA from switch i has been received; and F[i][p], a boolean flag indicating whether the switch has received via port p either the Seq[i]-th LSA from switch i or the corresponding dummy, for $1 \leq i \leq N$ and $1 \leq p \leq K_x$. Let us denote the sequence

[1]If necessary, LSAs received on the SMC can be identified by the VPI value of the MC.



(a) the broken SMC

(b) step 1: (partially failed) broadcast

(c) step 2: dummy forwarding

(d) step 3: real forwarding after timeout

(e) step 4: link-by-link forwarding triggered by real forwarding

Figure 3.10: The BE SAF protocol with a broken SMC.

number of LSA ℓ by Seq(ℓ) and the address of the source switch by Source(ℓ).

The source of an LSA invokes the routine **BE_SAF_Source**, which is shown in Figure 3.11. Parameters to the routine include the ID x of the invoking switch and an LSA ℓ to be flooded. The switch x updates the sequence number of its current LSA to that of ℓ and clears relevant F flags to indicate that it has not received anything about ℓ from its neighbors. The switch then broadcasts ℓ over the SMC, forwards the dummy of ℓ to all neighboring switches, and sets up a timer to await responses (for

ℓ or its dummies) from neighbors.

```
Algorithm: BE_SAF_Source.
Input: the switch ID $x$, and an LSA $\ell$.

Seq[$x$] = Seq($\ell$).
Received[$x$] = TRUE.
F[$x$][$p$] = FALSE, for all $1 \leq p \leq K$.
Transmit $\ell$ over the SMC.
Forward a dummy of $\ell$ to all neighboring switches.
Set up a timer($\ell$).
```

Figure 3.11: The sender algorithm of the BE SAF protocol.

Switches that receive an LSA ℓ invoke the routine BE_SAF_Receive, which is shown in Figure 3.12. The routine first decides whether it is dealing with a new LSA by checking ℓ's sequence number against the current sequence number recorded locally. If a new LSA is observed, corresponding F flags are cleared to indicate that nothing has yet been learned about this LSA from neighbors. The switch then checks whether the LSA arrived via the SMC or a point-to-point link. If the LSA arrived on the SMC, then its dummy is forwarded to all neighboring switches. Otherwise, the LSA itself is forwarded on all point-to-point links except the one on which it arrived. In the cases where dummies are forwarded, a timer is set up to make sure that the switch x hears responses from neighboring switches; the timeout handler is discussed later.

When a switch x receives the dummy of an LSA ℓ, it invokes the BE_SAF_Receive_Dummy routine, shown in Figure 3.13. The receipt of a dummy from a neighboring switch y assures x that y has already received the LSA and, therefore, that LSA forwarding to y is unnecessary. This situation is recorded in the corresponding F flags. As in the case of the BE_SAF_Receive routine, a check is made to determine if this is (the dummy of) a new LSA. If so, then the corresponding Seq entry is updated and relevant F flags are reset, as in the previous routine.

A switch x that receives an LSA ℓ from the SMC forwards the dummy, rather than the real LSA, to its neighboring switches. It also expects responses (ℓ itself or its dummy) from its neighbors. Lack of a response from a neighboring switch results in the forwarding of the real ℓ to that switch. The timeout-handler shown

```
Algorithm: BE_SAF_Receive.
Input: the switch ID $x$, and an LSA $\ell$ received from port $p$.

$y$ = Source($\ell$).
IF (Seq($\ell$) > Seq[$y$])
   Seq[$y$] = Seq($\ell$), and Received[$y$] = FALSE.
   F[$x$][$q$] = FALSE, for all $1 \leq q \leq K$.
ENDIF

IF (Received[$y$] is TRUE)
   Do nothing. /* drop LSAs that have been seen before */
ELSE
   /* This the first time this LSA is received */
   Received[$y$] = TRUE.
   IF ($p$ = 0) /* received from the SMC */
      Forward a dummy of $\ell$ to all neighboring switches.
      Set up a timer($\ell$).
   ELSE /* received from a point-to-point link */
      Forward $\ell$ on all point-to-point ports, except $p$.
   ENDIF
ENDIF
```

Figure 3.12: The receive-LSA routine in the BE SAF protocol.

```
Algorithm: BE_SAF_Receive_Dummy.
Input: the switch ID $x$, and a dummy $d$ received via port $p$.

$y$ = Source($d$).
IF (Seq($d$) > Seq[$y$])
   Seq[$y$] = Seq($d$), and Received[$y$] = FALSE.
   F[$y$][$q$] = FALSE, for all $1 \leq q \leq K$.
ENDIF
F[$y$][$p$] = TRUE.
```

Figure 3.13: The receive-dummy routine in the BE SAF protocol.

in Figure 3.14 checks the F flags to decide for all neighboring switches individually whether the forwarding of the real ℓ is necessary.

```
Algorithm: BE_SAF_Timeout_Handler.
Input:   the switch ID $x$ and a timer($\ell$).

IF (Seq[$\ell$] = Seq[Source($\ell$)])
   /* This timer is for the current LSA of the source of $\ell$.
      Out-of-date timers are ignored. */
   FOR (port number $p = 1$ to $K$) DO
         IF (F[Source($\ell$)][$p$] = FALSE)
            forward $\ell$ on port $p$.
         ENDIF
ENDIF
```

Figure 3.14: The timeout handler in the BE SAF protocol.

3.4 Performance Evaluation

The performance of the three alternative flooding methods (conventional flooding, Basic SAF, and BE SAF) is studied through simulation. The simulator is based on the CSIM package [48]. We are interested in both temporal and bandwidth metrics. Given a flooding operation and a switch x, the temporal metrics of the flooding operation include the time to receive an LSA, called the receipt time of x, and the time for a flooding operation to complete at x, called the completion time of x. (Completion time includes handling of duplicate LSAs and dummies.) In this study, we measured average receipt/completion times among all network switches. Confidence intervals were computed, but for most cases are very small and, for clarity, are not shown in plots.

The bandwidth consumption of a flooding operation can be measured by the number of links traversed by the LSA and, in the case of the BE SAF protocol, that of its dummy. The former number is denoted by B^a and the latter by B^d. Given length ℓ_a of an LSA and length ℓ_d of its dummy, the bandwidth consumed by a flooding operation is $B^a_f \times \ell_a + B^b_f \times \ell_b$, where $f \in$ {conventional, Basic SAF, BE SAF} is the method of the flooding. In this study, we obtained the B^a and B^d values through simulation runs, and we used the Fore SPANS NNI specification, where an n link-description LSA comprises $4 + 28 * n$ bytes [49], to determine the ℓ_a and ℓ_d values.

Networks comprising up to 256 switches were simulated; 20 graphs were generated randomly for each network size. Table 3.1 shows the characteristics of the graphs generated. In the table, a parenthesis in an entry represents the (minimum, average, maximum) triple of the corresponding metric. For example, in the case of the 20 4-node graphs, the minimum degree of a node, across all the graphs, was 1.0; the average minimum degree among the graphs was 1.35; and the largest minimum degree among the graphs was 2.0.

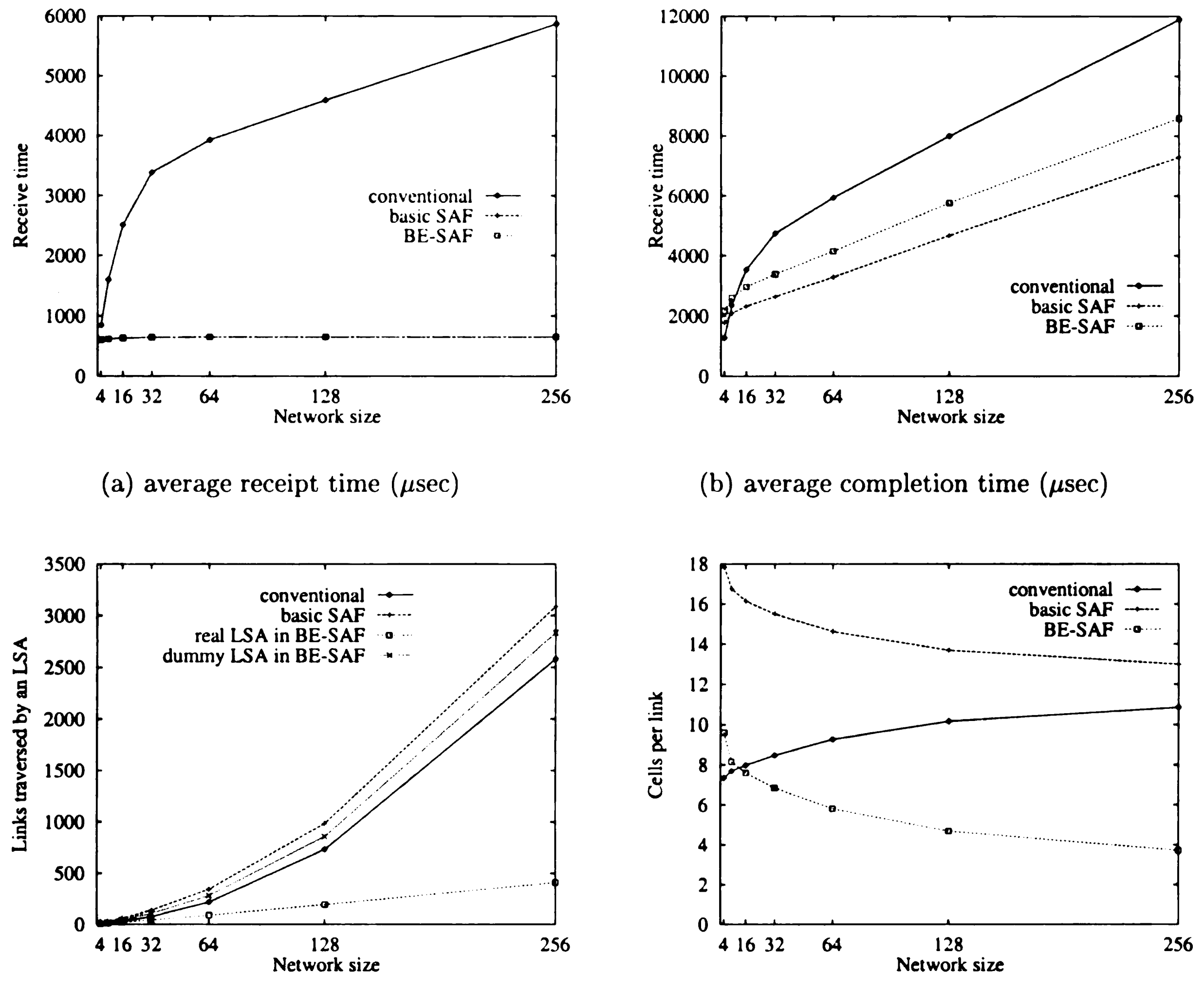
Size	min degree	max degree	avg degree	diameter
4	(1 1.35 2)	(2 2.70 3)	1.98	(2 2.200 3)
8	(1 1.10 2)	(3 3.95 5)	2.46	(3 3.850 6)
16	(1 1.00 1)	(4 5.25 7)	2.83	(4 5.550 9)
32	(1 1.00 1)	(5 7.05 9)	3.34	(5 6.700 12)
64	(1 1.10 2)	(6 9.40 12)	4.42	(4 6.150 11)
128	(1 1.50 3)	(9 13.95 19)	6.75	(4 5.400 9)
256	(1 3.60 7)	(11 21.30 29)	11.14	(4 4.800 8)

Table 3.1: Characteristics of randomly generated graphs.

Each communication operation, such as message forwarding, incurs ATM protocol overhead. We measured these overheads 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.

The final simulation parameter is the duration of the timer used by the BE SAF protocol, which is used in awaiting responses from neighboring switches. In the simulations, we set the timer according to the average degree of the given network graph G. Specifically, we set the timer to be of length $AvgD(G) \times \alpha$, where α is the time to forward an LSA via a point-to-point link.

Experiment 1: Ideal cases. By ideal, we refer to situations in which the SMC is completely operational during flooding operations. Such is the case for periodic flooding during "normal" periods, when all network components function properly, and during the flooding of event LSAs when none of the faulty network components affect the operation of the SMC. The simulation results pertaining to this setting are plotted in Figure 3.15. As shown in Figures 3.15(a) and (b), the two SAF protocols offer significant advantage in both LSA receipt time and flooding completion time, due to their use of the SMC as a "short cut" broadcast medium.



(a) average receipt time (μsec)

(b) average completion time (μsec)

(c) number of links traversed

(d) bandwidth consumption

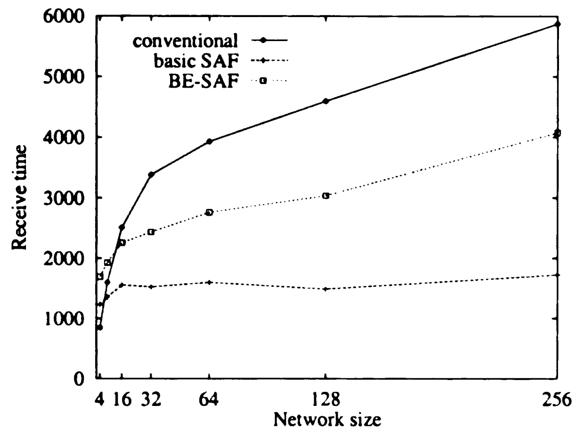
Figure 3.15: Comparisons of flooding alternatives with a correctly functioning SMC.

Figure 3.15(c) plots the number of links traversed by a real LSA or its dummy for the three protocols. As predicted by earlier analysis, the Basic SAF protocol consumes more bandwidth than does the conventional flooding algorithm; both methods do not use dummies, but the latter has smaller B^a values. With the use of dummy forwarding, the B^a values of the BE SAF protocol are significantly less than that of the conventional protocol, especially when the network is large. The actual bandwidth savings of the BE SAF protocol depends on the ℓ_a to ℓ_d ratio. Figure 3.15(d) plots the number of cells per link incurred by the three alternatives, assuming the use of Fore SPANS NNI and the inclusion of 10 link-state descriptions in the LSA.[2] Somewhat surprisingly, the BE SAF protocol consumes more bandwidth than do the other alternatives when the network is small (for example, containing fewer than than 8 switches). A closer examination of simulation runs reveals that this phenomenon is due to the small average degrees in small networks, leading to premature timer firings, followed by unnecessary LSA forwardings. This problem can be fixed by introducing longer timeout periods for small networks. For larger networks, the bandwidth savings of the BE SAF protocol are substantial.

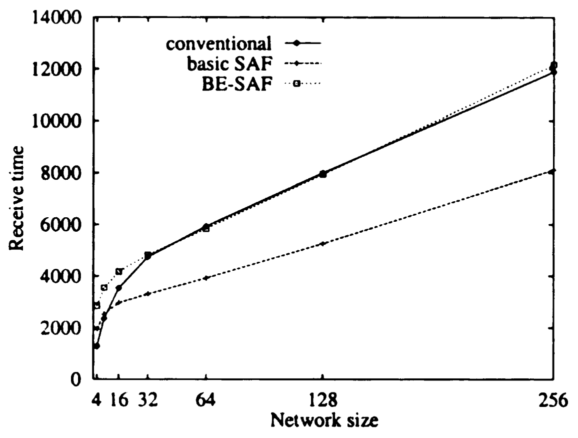
Experiment 2: Partitioned spanning MC. Next, we investigate the performance of the flooding algorithms when a (random) link used in the SMC fails, partitioning the MC into two segments. In this case, the two SAF protocols still use the MC to reach as many nodes as possible, and resort to link-by-link forwarding to reach the remaining nodes. The results are presented in Figure 3.16. As expected, the average receipt times for the two SAF protocols are larger than those of the previous experiment. However, they are still significantly smaller than those of the conventional flooding protocol. Provided that the component failure rate of the network is low, the occurrences of simultaneous failures should be rare. We suspect that the results of this experiment, which represent single-failure situations, combined with those of the previous experiment, which represent zero-failure situations, cover a very large fraction of network flooding situations.

Interestingly, the bandwidth consumption problem with the Basic SAF protocol diminishes with a partitioned SMC, as shown in Figure 3.16(d). This is because LSA broadcasts do not traverse all links of the partitioned SMC. On the other hand, the BE SAF protocol in this situation must forward real LSAs, rather than dummies,

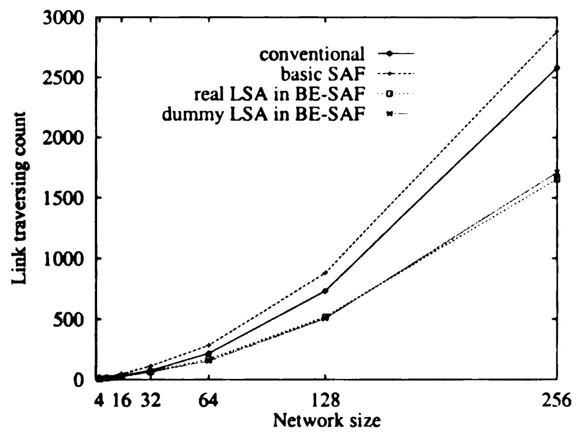
[2]An LSA must account for the links to neighboring switches as well as the links to hosts. Considering that most popular ATM switches can accommodate at least 16 ports, and some even 96 ports, we believe that 10 incident links per switch may be a relatively conservative representative figure.



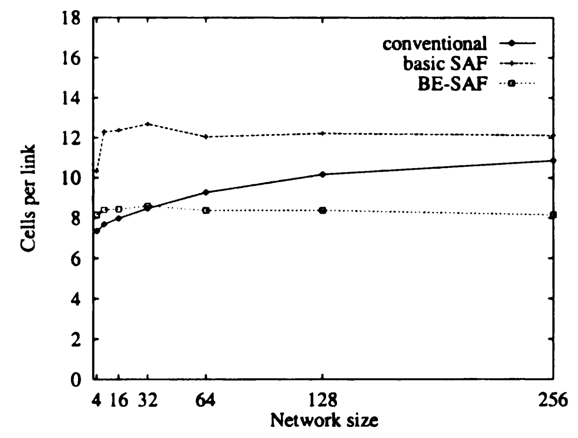
(a) average receipt time (μsec)



(b) average completion time (μsec)



(c) number of links traversed

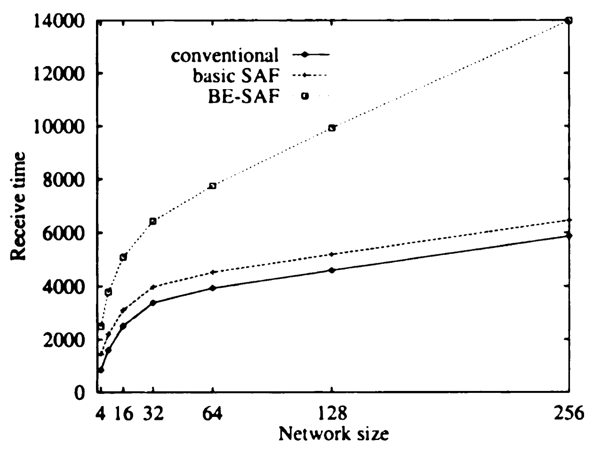


(d) bandwidth consumption

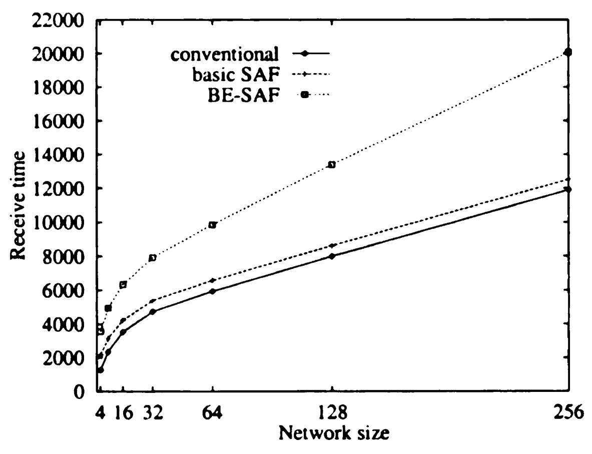
Figure 3.16: Comparisons of flooding alternatives with partitioned SMC.

in order to reach switches that are not covered by the SMC. Hence, the protocol consumes more bandwidth than it does in the previous experiment. However, its bandwidth consumption is still lower than that of the other alternatives.

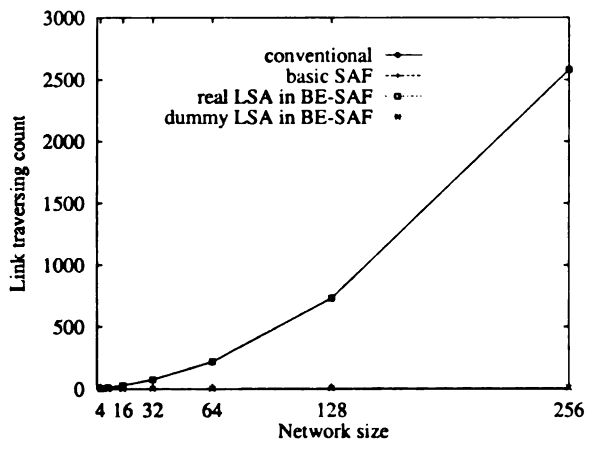
Experiment 3: Non-existent spanning MC. Let us now consider the worst-case setting for the SAF protocols: when the SMC does not exist at all. This situation may happen after network re-initialization and prior to reconstruction of the SMC, and can also be considered as a worst-case situation with respect to multiple link failures that partition the SMC. As we can see in Figure 3.17, the conventional flooding protocol outperforms the two SAF protocols in receipt time and completion time. The time differences between the conventional flooding protocol and the Basic SAF protocol are marginal, but those between the conventional flooding protocol and the BE SAF protocol are much more significant. The BE SAF protocol suffers in this experiment because each switch has to perform two rounds of forwarding to neighbors, one for dummies and one of real LSAs. In this experiment, the three flooding alternatives consume essentially the same amount of network bandwidth, because the two SAF protocols, like the conventional flooding protocol, use only link-by-link forwarding when the SMC does not exist.



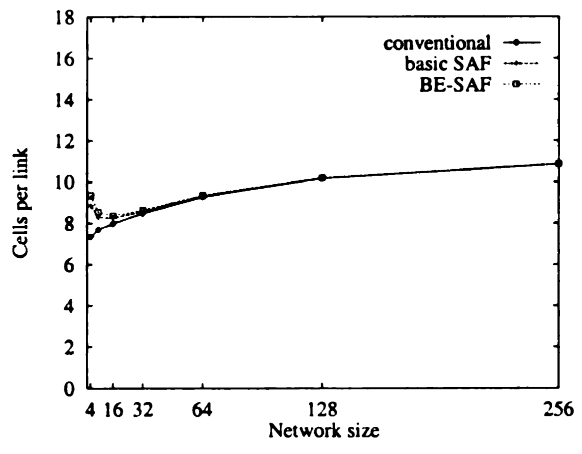
(a) average receipt time (μsec)



(b) average completion time (μsec)



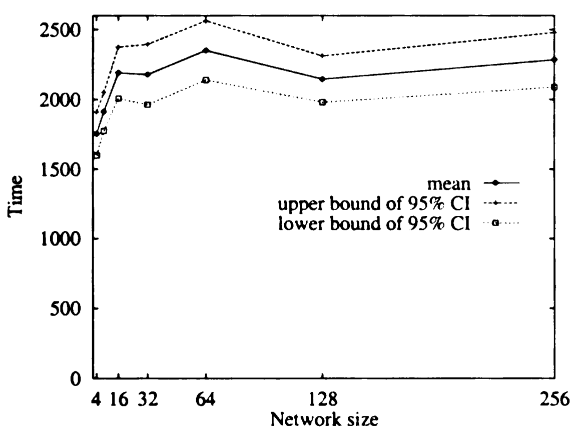
(c) number of links traversed



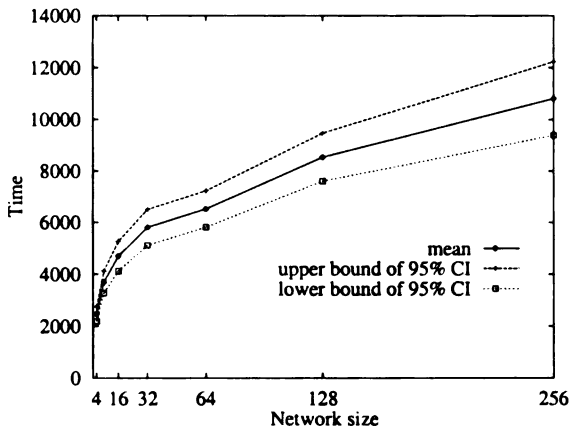
(d) bandwidth consumption

Figure 3.17: Comparisons of flooding alternatives when SMC does not exist.

Experiment 4: Performance of the SMC protocol. We also studied the performance of the SMC protocol under two scenarios: reorganization of an existing SMC when an SMC link fails, and construction of a new SMC. Corresponding simulation results, along with confidence intervals, are plotted in Figure 3.18. The results show that a partitioned SMC requires less than 2.5 milliseconds to reorganize, while constructing an SMC from scratch (a relatively rare event) requires less than 12 ms.



(a) time to reorganize a partitioned SMC (μsec)



(b) time to construct a new SMC (μsec)

Figure 3.18: Performance of the SMC protocol.

Interpretation of results. Among the three flooding alternatives, the BE SAF protocol experiences the most variation in performance across the three experiments. The ideal setting for the protocol occurs when there are no event LSAs and network switches simply flood status information periodically. The LSAs in this setting tend to be long because periodic flooding must include descriptions for all incident links, including those connected to hosts. In Experiment 1, the BE SAF protocol is fast and consumes significantly less bandwidth than do the other two alternatives. The protocol also favors long LSAs for the effectiveness of dummy forwarding. Besides the bandwidth benefit, the use of dummy forwarding might also reduce ATM protocol overhead during link-by-link forwarding; researchers have reported that the ATM protocol overhead of one-cell packets (such as LSA dummies) can be dramatically reduced if these packets are treated as a special case [50]. We conclude that the BE SAF protocol is the best choice for periodic flooding during normal network operation. On the other hand, the worst-case behavior of the BE SAF protocol is worst among the three flooding alternatives. However, the adverse scenarios considered in Experiments 2 and 3 are likely to stem from emergency events, such as component failures, whose LSAs are typically short. Given these results, we may conclude that a good heuristic would be to use the BE SAF protocol for periodic flooding or fluctuation in resource availability (for example, changes in residual bandwidth of a link), but to invoke the Basic SAF protocol for the dissemination of network component failures.

3.5 Summary

In this chapter, we have proposed two switch-aided flooding protocols and an accompanying protocol to construct spanning MCs. The protocols are designed to exploit ATM hardware cell switching and cell duplication. SAF protocols use the SMC as a broadcast medium to reduce flooding time. However, the protocols do not rely entirely on the SMC, but rather revert to point-to-point message forwarding if the SMC is damaged or under construction.

Two SAF protocols were described: the Basic SAF protocol and the BE SAF protocol. Under normal operating conditions, both protocols deliver network updates several times faster than the conventional flooding algorithm. The Basic SAF protocol is a relatively simple extension of the conventional flooding protocol and should be straightforward to implement. Our simulation study shows that the difference in the bandwidth consumed by the Basic SAF protocol and the conventional flooding is significant for small networks, but is only marginal for large networks. The advantage of the Basic SAF protocol over the BE SAF protocol is its stability in performance under adverse circumstances, for example, when the SMC is partitioned or under construction. We also note that the bandwidth consumption of this protocol may be even smaller when flooding event LSAs, due to their short lengths; under the Fore SPANS implementation, event LSAs are one-cell packets.

The BE SAF protocol addresses the bandwidth consumption issue by introducing dummy LSA forwarding. The bandwidth savings of this method is particularly significant when the network size is large or when the LSAs are long. The performance of the BE SAF protocol is more sensitive to adverse network circumstances, however. As a simple heuristic, an "adaptive" network management system could use the BE SAF protocol for periodic flooding operations (whose corresponding LSAs are typically sufficiently long to benefit from the use of dummy forwarding), but switch to the Basic SAF protocol in the presence of emergency events, such as link failures.

The results in this chapter support the theme of this dissertation: the mutual beneficial relation between LSR and group communication. We have demonstrated that group communication techniques help improve the performance of LSR. Specifically, we have used a spanning tree to improve the performance of flooding operations. In the next chapter, we will push farther in this direction, introducing another type of topology that has been used in host-level group communication, the ring topology, to further improve flooding performance. We will show that the combination of a spanning tree and a ring produces an optimal flooding method for use by ATM networks.

Chapter 4

Optimal SAF Operations

In the previous chapter, we improved the performance of flooding operations in ATM networks by constructing a tree topology, a common technique of supporting group communication. Another type of topology that has been used in group communication is the ring topology, which connects the members of a group in a circular manner. Host-level applications of a "group ring" include barrier synchronization [51], leader election [52], reliable multicast [53], and maintaining among group members consistent orderings of receiving messages [53]. In this chapter, we construct a ring that uses ATM VCs to connect all switches in an ATM network for use as the acknowledgment topology in flooding operations. Switches, after receiving an LSA from the SMC, exchange acknowledgments or dummy LSAs only with neighboring switches defined by the ring, as opposed to all the neighboring switches defined by the physical topology. The resultant flooding protocol, called Efficient Reliable (ER) SAF, is optimal in terms of complexity, for it requires only $O(1)$ complexity in LSA receipt time and flooding completion time, and incurs only $O(|V|)$ bandwidth for both LSA delivery and reliability implementation.

4.1 Motivation

In the previous chapter, we developed two SAF methods, namely the Basic SAF and BE SAF protocols. These two SAF protocols outperform the conventional flooding algorithm by using a hardware-based spanning tree, the SMC, to speed up the dissemination of LSAs. We note that the (remaining) overheads of the two SAF protocols stem from the requirement to guarantee the delivery of any LSA to all network nodes that are reachable from the originating node. In general, both the conventional flooding method and previous SAF protocols achieve reliability by means of a "neighbor watching" principle: every node, after receiving an LSA, makes sure that all its neighboring nodes have also received the LSA. In the conventional flooding protocol, the principle is implemented by reliably forwarding an incoming LSA to all neighbors, except the one from which the LSA arrives. In the Basic SAF and BE SAF protocols, the principle is implemented by exchanging acknowledgments or dummy LSAs with all neighboring switches. The communication of every switch with all neighboring switches inevitably consumes $O(|E|)$ bandwidth and requires $O(D_G)$ time to complete a flooding operation, where D_G is the maximum node degree in the given network topology G. To avoid overheads that are associated with reliability, one could use the SMC for best-effort flooding and ignore the reliability issue altogether. In this method, which we refer to as the *Unreliable SAF* protocol, the source node of an LSA broadcasts the LSA on the SMC, but makes no effort to ensure receipt of the LSA by other switches.

The speed and bandwidth complexities of the four flooding protocols discussed so far (the conventional, Basic SAF, BE SAF, and Unreliable SAF protocols) are compared in Table 4.1, where dia_G denotes the diameter of network G. In the table, we distinguish two bandwidth metrics: *delivery bandwidth* refers to the number of links that an LSA has to traverse, and *reliability bandwidth* refers to the number of acknowledgments/dummies produced. As we can see in the table, the three SAF protocols are more efficient than the conventional flooding protocol. The Unreliable SAF protocol is the most efficient, of course, since it does not include acknowledgments: it exhibits constant complexities in both time metrics and consumes $O(|V|)$ bandwidth.

Of course, we would like to use the most efficient flooding protocol available. One method to use the Unreliable SAF protocol is to distinguish two types of network status: topology status and utilization status. As discussed earlier, the *topology status*

Table 4.1: Complexities of various flooding protocols.

Flooding Method	Time		Bandwidth		
	Receipt	Completion	Delivery	Reliability	Total
Conventional	$O(dia(G))$	$O(dia(G) + deg_G)$	$O(\|E\|)$	$O(\|E\|)$	$O(\|E\|)$
Basic SAF	$O(1)$	$O(deg_G)$	$O(\|E\|)$	$O(\|E\|)$	$O(\|E\|)$
BE SAF	$O(1)$	$O(deg_G)$	$O(\|V\|)$	$O(\|E\|)$	$O(\|E\|)$
Unreliable SAF	$O(1)$	$O(1)$	$O(\|V\|)$	0	$O(\|V\|)$
ER SAF (this chapter)	$O(1)$	$O(1)$	$O(\|V\|)$	$O(\|V\|)$	$O(\|V\|)$

of a network component (a switch or a communication link) refers to the operational state of the component; the present topology of the network is determined by the set of currently operational switches and links. The topology of a network can be expected to be relatively static, assuming that reliable components are used to construct the network. The *utilization status* reflects the availability of network resources. For example, the utilization status of a link includes the bandwidth in use, the delay over the link experienced by recent cells, cell loss rate, and so forth. In ATM networks, utilization status could be very dynamic, as network resources are allocated and released when VCs are set up and torn down. As such, utilization status LSAs are expected to constitute the majority of flooding operations.

It has been argued [54, 55] that, while changes in topology status (such as the failures of network components) must be flooded reliably, dynamics in utilization status could use unreliable, or best-effort, flooding methods. This is because inaccurate resource utilization information would not lead to disastrous situations, but merely result in sub-optimal routing decisions. Moreover, since the utilization of network resources may change at a high rate, one should be concerned with the *efficiency* of disseminating such changes. It follows that the Unreliable SAF protocol best fits this purpose. We agree that efficiency is a major concern in the flooding of utilization status LSAs. However, in this chapter we will demonstrate that a reliable SAF protocol *can* be complexity-wise as efficient as the unreliable SAF protocol. Furthermore, we contend that there are cases where the reliability of resource utilization flooding *is* important.

Let us consider a switch x that has been overloaded by heavy traffic. According to ATM PNNI, at least one LSA indicating the utilization change will be flooded throughout the network so that other switches can avoid using switch x in future VCs. Should switch x advertise the congestion situation unreliably, some switches may not receive the corresponding LSA and thus will continue using the switch in new VCs, further exacerbating the congestion situation. Moreover, it is exactly when a switch is congested that it will most likely drop cells, including the ones pertaining to utilization status LSAs that disseminate the congestion situation. The information about the congestion at x may not leave x at all, and the problem feeds on itself as new VCs make the congestion situation worse.

In this chapter, we continue the SAF work by developing a reliable SAF protocol that is more efficient than the Basic and BE SAF protocols. The Efficient Reliable (ER) SAF protocol constructs a second topology, a virtual ring, to provide reliability. As shown in Table 4.1, the new protocol exhibits speed and bandwidth complexities identical to those of the unreliable SAF protocol. Further, it retains the reliability of the conventional flooding protocol, that is, an LSA will be delivered to all switches that are reachable from the originating switch. Since a flooding protocol must deliver a given LSA at least once to every such switch, both the $O(1)$ time complexities and the $O(|V|)$ bandwidth complexities of the ER SAF protocol are optimal.

The remainder of this chapter is organized as follows. In Section 4.2, we describe the ER SAF protocol, including the use of the virtual ring for reliability and issues that arise when decoupling construction/maintenance of the ring from on-going flooding operations. Details of the ER SAF algorithms are provided in Section 4.3. In Section 4.4, we discuss the methods used to construct and maintain the virtual ring. While the ER SAF protocol achieves optimal complexities, the expected performance under real network conditions is of interest. In Section 4.5, we investigate through simulation the behavior of the ER SAF protocol both in "normal" situations and under adverse circumstances, where network component failures affect the operation of the SMC and/or the virtual ring. The results of our simulation reveal that the ER SAF protocol delivers network updates several times faster than conventional approaches in normal situations, and twice as fast in the presence of component failures.

A summarization of our SAF work is given in Section 4.6.

4.2 ER SAF Protocol Design

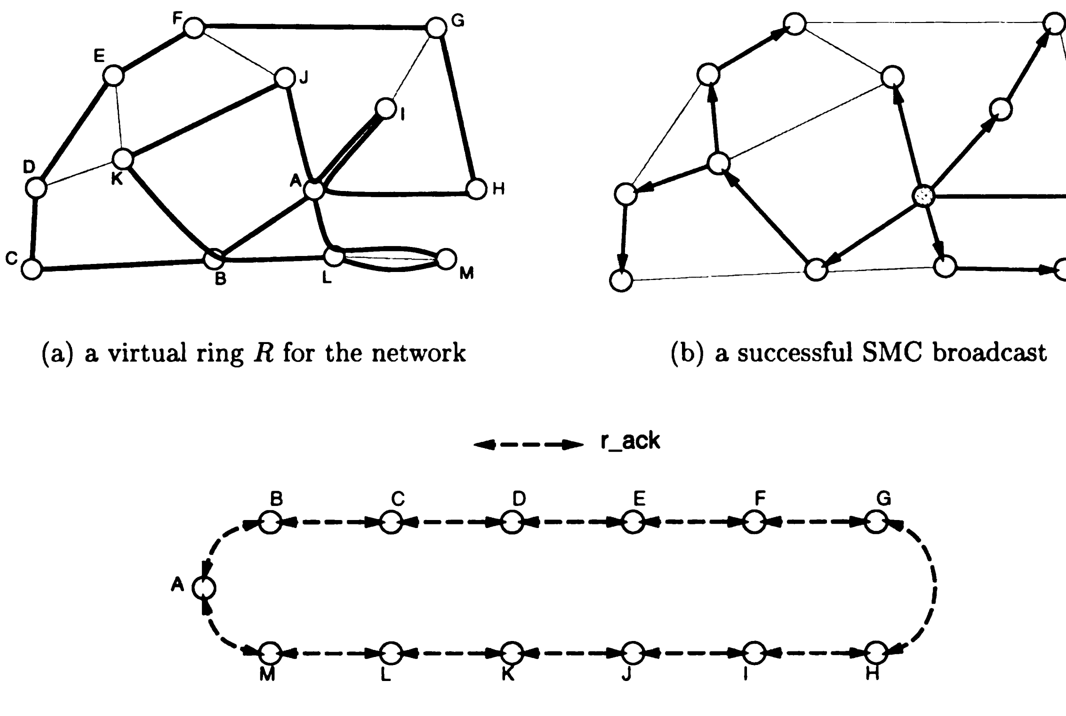
In this section, we describe the design issues and basic concepts of the ER SAF protocol. In the discussion, we assume that the network topology $G = (V, E)$ is a connected graph, since our concern here is to efficiently flood LSAs to "reachable" nodes. To generalize our discussion to partitioned networks, we can simply apply the argument to each segment.

4.2.1 Basic Concept

The ER SAF protocol uses the hardware-based SMC to achieve constant LSA delivery time. However, it adopts a different approach to reliability than previous SAF protocols. Instead of implementing the neighbor watching principle over the physical network topology G, the ER SAF protocol constructs a virtual topology $R = (V, E_R)$ to implement reliability. The topology R is a ring that visits all nodes in G exactly once. The topology is *virtual* because neighboring nodes in R are not necessarily adjacent in the physical network topology G. Rather, they are connected by ATM VCs that may traverse one or more intermediate nodes. Specifically, each node x in G is connected to its predecessor in R, denoted as Pred(x), and to its successor in R, denoted as Succ(x), by VCs $\text{RVC}_{\text{pred}}(x)$ and $\text{RVC}_{\text{succ}}(x)$, respectively. (RVC stands for Ring VC.) We defer to Section 4.4 the discussion of the construction and maintenance of ring R. At this point, we merely emphasize that the ER SAF protocol must be able to work properly when the ring R is under construction or involved in maintenance operations.

In the ER SAF protocol, the neighbor watching principle is implemented as follows. Any node x, after receiving an LSA from the SMC, exchanges acknowledgments of the LSA with Pred(x) and Succ(x), rather than with all its neighboring nodes defined by the physical topology G; we will refer to acknowledgments sent via ring VCs as *r_acks*. If every node $x \in G$ receives r_acks for a given LSA from Pred(x) and

Succ(x), then the flooding operation is completed. Let us use an example to illustrate the operation of the ER SAF protocol in "normal" cases, where the topology of the network is stable and both the SMC and virtual ring R are fully operational. This example assumes the network and SMC topologies shown in Figure 3.1(c). Figure 4.1(a) depicts a virtual ring R connecting switches in the (alphabetic) order $A, B, C, \ldots, M$. We point out that some ring VCs, such as the H-I VC, traverse one or more intermediate nodes. Assuming that the SMC broadcast of an LSA ℓ successfully reaches all switches, as shown in Figure 4.1(b), ensuing neighbor watching activities are depicted in Figure 4.1(c), where each node exchanges r_acks of ℓ with its succeeding and preceding nodes in R. The flooding operation is completed when every node receives two acknowledgments of ℓ.



(a) a virtual ring R for the network

(b) a successful SMC broadcast

(c) exchange of acknowledgments in R

Figure 4.1: ER SAF flooding in normal cases.

In ER SAF operations under normal conditions, nodes require $O(1)$ time to receive the LSA, and must process $O(1)$ r_acks. Hence, the per switch workload (that is, the completion time metric) is of constant complexity. $O(|V|)$ acknowledgments will be produced; the total number of links traversed by r_acks depends on the total length of ring VCs, denoted as $|R|$. Various existing heuristics for the traveling salesman problem produce cycles where $|R| < C \times |V|$ and C is a constant [56]. Using such a heuristic in the construction of the ring, the bandwidth consumed by reliability activities is of complexity $O(|V|)$ (our simulation results presented in Section 4.5 show that C is typically less than 1.5). Because the number of links that an LSA traverses in normal cases is exactly the number of SMC links, the bandwidth consumed by LSA delivery also exhibits complexity $O(|V|)$. Thus, the total bandwidth consumption exhibits complexity $O(|V|)$.

4.2.2 Operation Modes

In addition to normal situations described above, the ER SAF protocol must handle more difficult scenarios where the SMC broadcast of the LSA does not reach all nodes, where cells pertaining to r_acks are lost, where ring VCs are damaged by network component failures, or where arbitrary combinations of these events occur. If an LSA is being flooded under such adverse circumstances, then there may exist a node x that possesses the LSA after the SMC broadcast but does not receive the r_ack of the LSA from a node $y \in \{\text{Pred}(x), \text{Succ}(x)\}$. (If no such node x exists, then the flooding is completed.) In this case, node x can retransmit the LSA to y using the corresponding ring VC, and repeat such retransmissions until y returns an r_ack. For a given LSA, when the ER SAF protocol uses the virtual ring R for acknowledgments/retransmissions, we say that it is operating in the R *mode*.

Adverse network status changes and the R-mode operation create a cyclic dependency: in mode R, adverse network changes that damage the ring R can impede their own advertisements, while the repair of the ring R requires up-to-date network topology information contained in such advertisements. To avoid this dilemma, the ER SAF protocol has a second operation mode, called the G *mode*, that is used when the ring R is damaged or under construction (the letter G indicates the use of the physical topology G for reliability). When operating in the G mode, the ER SAF protocol is identical to the Basic SAF protocol: a node receiving the LSA on the

SMC subsequently exchanges copies of the LSA (and acknowledgments) with each of its physical neighbors.

The ER SAF protocol needs a method to decide which mode to use for a given LSA. In general, the R mode, due to its efficiency, should be used whenever it can ensure reliability, that is, when the ring R is operational; otherwise, the G mode should be used. The source node a of an LSA uses its "local" status of R, that is, the operational status of $\mathrm{RVC}_{\mathrm{succ}}(a)$ and $\mathrm{RVC}_{\mathrm{pred}}(a)$, to determine the mode to use for the LSA. If both RVCs are operational, then the source node initiates the flooding operation in mode R. Otherwise, it starts the flooding in mode G.

Of course, it is possible that the source of an LSA initiates a flooding operation in the R mode, while there are link-down events that damage ring R and that have not yet been learned of by the source. In such circumstances, some node(s) other than the source must change the operation mode during the course of the flooding. In ER SAF, the flooding of a given LSA can change from the R mode to the G mode, but the reverse is not allowed. Consider a scenario where a switch a is flooding a utilization status LSA ℓ using mode R, while in the meantime a link used by $\mathrm{RVC}(x, y)$ but not by the SMC has failed. Let us assume that the SMC broadcast of ℓ successfully arrives at all nodes in G. Although both x and y receive ℓ from the SMC, the two nodes cannot receive r_acks of ℓ from one another. Both nodes will try to retransmit ℓ to each other, but such retransmissions have no chance to succeed either. The R-mode flooding operation is bound to fail in this situation. Instead, node x, after realizing the problem with the $\mathrm{RVC}(x, y)$, must switch to mode G, initiating a basic SAF operation of ℓ on behalf of switch a. (Node x could learn of the problem via the link-down LSA produced by the endpoints of the faulty link or when retransmissions fail a predetermined number of times.) In this manner, we are assured that ℓ will reach all network nodes while the ring R is under repair.

Even when the ring R is fully operational, there are cases where the R mode is unacceptably inefficient. Consider the example shown in Figure 4.2, where switch G, which is a leaf in the SMC, is advertising the failure of the (G,I) link, which is used by the SMC, but not by the virtual ring R (we assume the ring topology depicted in Figure 4.1(a)). In this situation, the SMC cannot deliver this link-down LSA to any node at all. After failing to receive corresponding r_acks from Pred(G)=F and Succ(G)=H, switch G retransmits the LSA to the two nodes over ring VCs. Nodes F and H will also notice the lack of r_acks from E and I, respectively, and attempt to retransmit. The result is that the LSA traverses the ring R in a sequential, store-and-forward manner, as depicted in Figure 4.2(b). In general, retransmissions over the ring R degenerate into a sequential procedure whenever multiple nodes, consecutive in R, fail to receive the SMC-switched copy of an LSA. To avoid this performance problem, we introduce a *two-three rule* as a mode-switching heuristic: whenever any two consecutive nodes in R do not receive a given LSA from the SMC, the ER SAF operation, with respect to that LSA, will switch to mode G. This rule can be formally stated as follows.

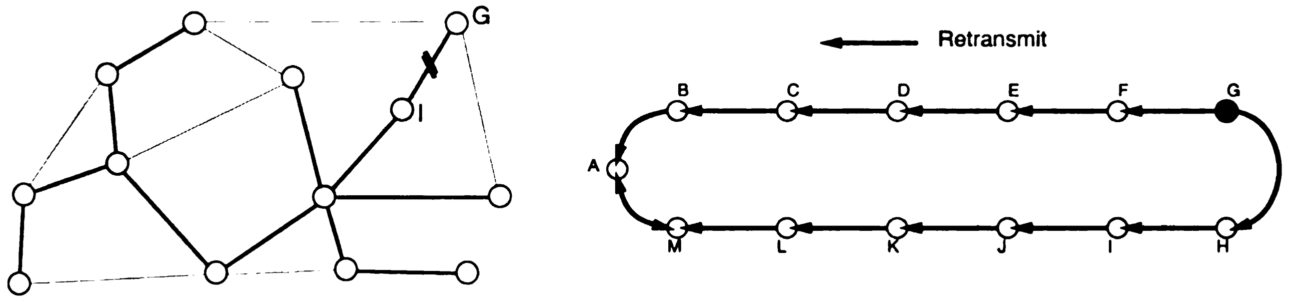
Two-Three Rule. With respect to a given LSA ℓ, the two-three rule is satisfied at a node x if any two of the three nodes, x, Succ(x), and Pred(x), do not receive ℓ from the spanning MC. Precisely, any node x that is currently in mode R with respect to ℓ switches to mode G if either one of the following conditions is satisfied.

C1. Node x does not receive an r_ack of ℓ from either Succ(x) and Pred(x) after waiting for a predetermined length of time since the receipt of SMC-relayed copy of ℓ.

C2. The first time x receives the LSA is from one of its ring neighbors (indicating x itself has missed the SMC copy), but has not received the r_ack from the other ring neighbor y (indicating that y may not receive the LSA either).

4.3 Algorithms

In this section, we present the algorithms used by the ER SAF protocol. We use the notation $N_R(x)$ to denote the set {Succ(x), Pred(x)} and the notation $N_G(x)$



(a) the link (H, J) fails (b) sequential store-and-forward in R

Figure 4.2: A hypothetical scenario where LSA retransmissions over R degenerate into a bidirectional store-and-forward process.

to denote the set of neighboring nodes defined by the network topology G. We assume that nodes in $N_G(x)$ can be reached via ports numbered 1 to K_x, where $K_x = |N_G(x)|$. We further assume that LSAs are tagged with a sequence number and source switch ID: an LSA from node i with sequence number j will be denoted as LSA(j,i), and its corresponding acknowledgments will be denoted as either r_ack(j, i) or g_ack(j, i), depending on the operation mode of the LSA. Each switch maintains the following data structures: Seq[i], the sequence number of the current LSA from switch i, $1 \leq i \leq |V|$; Mode[i] $\in \{\mathsf{R}, \mathsf{G}\}$, the operation mode for the flooding of LSA(Seq[i],i); $\mathrm{F}_{\mathrm{succ}}[i]$ and $\mathrm{F}_{\mathrm{pred}}[i]$, two boolean flags indicating whether x has received r_ack(Seq[i],i)/LSA(Seq[i],i) from nodes Succ(x) and Pred(x) respectively; and F[i][p], a boolean flag indicating whether the switch has received via port p the g_ack(Seq[i],i)/LSA(Seq[i],i), for $1 \leq i \leq |V|$ and $1 \leq p \leq K_x$. Let us denote the sequence number of LSA ℓ by Seq(ℓ) and the address of the source switch by Source(ℓ). Moreover, every LSA ℓ has a mode bit, denoted by Mode(ℓ), whose value can be either R (for mode R) or G (for mode G). The mode bit of an LSA may change value during the course of the corresponding flooding operation, and copies of the same LSA may be in different modes. We also assume that, when a switch x receives a copy of ℓ, it can discover the sender of this copy, denoted as Sender(ℓ). The sender information can be determined by the port/RVC/SMC on which ℓ arrives.

The source of an LSA invokes the routine **FloodLSA**, which is shown in Figure 4.3.

Parameters to the routine include the ID x of the invoking switch and an LSA ℓ to be flooded. The switch x updates the sequence number of its current LSA to that of ℓ and clears relevant F_{succ}, F_{pred}, and F flags to indicate that it has not received anything about ℓ from its neighbors. Finishing these bookkeeping tasks, switch x next decides the operation mode of ℓ. The R mode is used if incident ring VCs, $RVC_{succ}(x)$ and $RVC_{pred}(x)$, are operational and if at least one incident SMC link is operational. (The use of mode R even when some incident SMC links are malfunctioning encourages the use of fragments of the SMC to disseminate an LSA to as many nodes as possible.) In this case, switch x sends the LSA on the SMC with mode bit set to R, and sends r_acks of the LSA to nodes in $N_R(x)$ via ring VCs. Otherwise, the G mode is used, that is, switch x sends the LSA on the SMC with mode bit set to G, and forwards the LSA to nodes in $N_G(x)$ via physical links. The SMC broadcast may be skipped in mode G, however, if all SMC links incident to x have failed. In either mode, a timer is setup to wait for responses from the set of neighboring nodes determined by the chosen operation mode.

```
Algorithm: FloodLSA.
Input: the switch ID $x$, and an LSA $\ell$.

Seq[$x$] = Seq($\ell$).
F[$x$][$p$] = FALSE, for all $1 \le p \le K$.
$F_{succ}[x] = F_{pred}[x]$ = FALSE.
IF (either $RVC_{succ}(x)$ or $RVC_{pred}(x)$ is damaged) or
   (all SMC links incident to $x$ are malfunctioning)
   Mode($\ell$) = Mode[$x$] = G.
ENDIF
IF (mode[$x$] = R)
   Broadcast $\ell$ on the SMC.
   Send r_ack(Seq($\ell$),$x$) to the two nodes in $N_R(x)$ via ring VCs.
   Set up an r_timer($\ell$).
ELSE /* mode[$x$] = G*/
   IF (at least one incident SMC link is operational),
      Broadcast $\ell$ on the SMC.
   ENDIF
   Forward $\ell$ to all nodes in $N_G(x)$ via physical links.
   Set up a g_timer($\ell$).
ENDIF
```

Figure 4.3: The sender algorithm of the ER SAF protocol.

Switches that receive an LSA invoke the routine **ReceiveLSA**, which is shown in Figure 4.4. Parameter x indicates the ID of the invoking switch and parameter ℓ is the LSA received. The routine first decides whether it is dealing with a new LSA by checking ℓ's sequence number against the current sequence number recorded locally. If a new LSA is observed, corresponding F_{succ}, F_{pred}, and F flags are cleared to indicate that nothing has yet been learned about this LSA from neighbors, and the network image at x is updated according to ℓ. Subsequent processing depends on the mode of ℓ. The processing of ℓ when Mode(ℓ)=G follows the Basic SMC protocol: if ℓ arrives on the SMC, then it is forwarded on all ports; otherwise, ℓ arrives on a port p and is forwarded on all the other ports. The processing of ℓ when Mode(ℓ)=R is somewhat complicated. First, the switch needs to decide whether to change to the G mode. If ℓ arrives on the SMC, its mode is changed when any incident RVC of x is damaged. If ℓ arrives on a ring VC, its mode is changed when condition C2 of the Two-Three Rule is satisfied. If mode switching does occur, ℓ is forwarded on all ports p, $1 \leq p \leq K_x$. Otherwise, the r_ack of ℓ is sent to Pred(x) and Succ(x) if ℓ arrived on the SMC, or to the Sender(ℓ) if ℓ arrived on a ring VC. This concludes the processing of ℓ when it is received for the first time. When switch x receives subsequent copies of ℓ, it discards such copies, unless the current mode at x with respect to ℓ is the R mode and the arriving copy is in mode G, forcing x to switch to the G mode and to forward ℓ along all incident links. We point out, again, that sending an acknowledgment is necessary even for duplicate copies of ℓ. Lastly, if ℓ did not arrive on the SMC, switch x must remember that it has received ℓ from its neighboring node Sender(ℓ).

When a switch x receives the acknowledgment of an LSA ℓ, it invokes the **ReceiveACK** routine, shown in Figure 4.5. The purpose of the routine is straightforward: The receipt of an acknowledgment from a switch y assures x that y has already received the LSA. This situation is recorded in the corresponding F_{succ}, F_{pred}, or F flags, according to the port/RVC on which the acknowledgment arrives.

When the timer associated with LSA ℓ fires at switch x, the switch invokes the **TimeoutHandler** routine shown in Figure 4.6. The timer, however, may be ignored for two reasons: it was set up for an LSA with an obsolete sequence number, or the

```
Algorithm: ReceiveLSA.
Input: the switch ID $x$, and an LSA $\ell$.

$a$ = Source($\ell$).
IF (Seq($\ell$) > Seq[$a$]) /* This is the first copy. */
    Seq[$a$] = Seq($\ell$), $F_{succ}[x] = F_{pred}[x]$ = FALSE, and F[$x$][$q$] = FALSE, for all $1 \leq q \leq K$.
    Update the local network image at $x$ according to $\ell$.
    IF (Mode($\ell$) = R)
        IF ($\ell$ is received from the SMC)
            IF ((both $RVC_{succ}(x)$ and $RVC_{pred}(x)$ are operational)
                Send r_ack(Seq($\ell$),$a$) via the two RVCs, and set up an r_timer($\ell$).
            ELSE
                Change Mode($\ell$) to G, forward $\ell$ on port $p$, $1 \leq p \leq K_x$, and set up a g_timer($\ell$).
            ENDIF
        ELSE /* $\ell$ is received from a ring VC $v$; check condition C2 of the Two-Three Rule */
            IF (Sender($\ell$)=Pred($x$) and $F_{succ}[x]$=FALSE) or
                (Sender($\ell$)=Succ($x$) and $F_{pred}[x]$=FALSE)
                Change Mode($\ell$) to G, forward $\ell$ on port $p$, $1 \leq p \leq K_x$, and set up a g_timer($\ell$).
            ELSE
                Send r_ack(Seq($\ell$), $a$) to Sender($\ell$) via RVC $v$.
            ENDIF
        ENDIF
    ELSE /* Mode($\ell$) = G */
        IF ($\ell$ is received from the SMC)
            Forward $\ell$ on port $p$, $1 \leq p \leq K_x$, and set up a g_timer($\ell$).
        ELSE /* $\ell$ is received from port $p$ */
            Forward $\ell$ on all ports, except $p$, and set up a g_timer($\ell$).
            Send g_ack(Seq($\ell$), $a$) to Sender($\ell$) via port $p$.
        ENDIF
    ENDIF
    Mode[$a$] = Mode($\ell$).
ELSE /* This is an extra copy. */
    If (Mode[$a$] = R) but (Mode($\ell$) = G)
        Change Mode[$a$] to G, forward $\ell$ on port $p$, $1 \leq p \leq K_x$, and set up a g_timer($\ell$).
    ELSE
        Return an r_ack or g_ack to the Sender($\ell$), depending on Mode($\ell$).
    ENDIF
ENDIF
IF ($\ell$ is received from a ring VC)
    Set $F_{pred}[a]$ or $F_{succ}[a]$ to TRUE, depending on Sender($\ell$).
ELSE IF ($\ell$ is received from port $p$)
    F[a][p] = TRUE.
ENDIF /* No flag to set if $\ell$ is received from the SMC. */
```

Figure 4.4: The **ReceiveLSA** routine in the ER SAF protocol.

```
Algorithm: ReceiveACK.
Input: the switch ID $x$, and an acknowledgment $d$.

$a$ = Source($d$).
IF ($d$ is received from Pred($x$))
    $F_{\text{pred}}[a]$ = TRUE.
ELSE IF ($d$ is received from Succ($x$))
    $F_{\text{succ}}[a]$ = TRUE.
ELSE /* $d$ is received from port $p$ */
    $F[a][p]$ = TRUE.
ENDIF
```

Figure 4.5: The ReceiveACK routine in the ER SAF protocol.

timer is an r_timer for an LSA whose operation mode at x has been changed to mode G since the setup of the timer. Subsequent processing, if required, depends on the type of the timer. For a g_timer, the routine forwards the associated LSA ℓ to ports whose corresponding F flags have not been set to TRUE. The processing of an r_timer is more complicated, however, as we have to decide whether to switch to mode G. The mode is changed when local RVCs are found to be damaged, or when condition C1 of the Two-Three Rule is satisfied. If the mode needs to be changed, then the LSA is forwarded on all ports p, $1 \leq p \leq K_x$. Otherwise, the LSA is forwarded (or, more precisely in this case, retransmitted) to a node in $N_R(x)$ whose corresponding flag is FALSE (at most one such retransmission will be performed, otherwise the Two-Three Rule would have been satisfied). Lastly, a timer in an appropriate mode is set up to wait for responses from neighboring nodes to which this LSA has been forwarded/retransmitted.

4.4 The Virtual Ring

In this section, we discuss the construction and maintenance of the virtual ring R. For this topic we must explicitly consider the handling of network partitioning, and hence we drop the assumption that the network G is connected. While a flooding operation is only concerned with delivering the LSA to all nodes reachable from the source node, the ring construction procedures of the ER SAF protocol must construct

```
Algorithm: TimeoutHandler.
Input: the switch ID $x$ and a timer($\ell$).

$a$ = Source($\ell$).
IF (Seq($\ell$) < Seq[$a$]) or (timer($\ell$) is an r_timer but Mode[$a$]=G)
   Return.
ENDIF

IF (timer($\ell$) is a g_timer)
   For $1 \le p \le K$, forward $\ell$ on port $p$ if (F[a][$p$] = FALSE).
   Set up a new g_timer($\ell$).
ELSE /* timer($\ell$) is an r_timer */
   IF (either $RVC_{succ}(x)$ or $RVC_{pred}(x)$ is damaged) or
      /* The next condition is the C1 of the two-three rule. */
      ($F_{succ}[a]$=FALSE and $F_{pred}[a]$=FALSE),
      /* Switch to the G mode. */
      Mode[$a$] = Mode($\ell$) = G.
      Forward $\ell$ via all incident port $p$.
      Set up a new g_timer($\ell$).
   ELSE
      /* Retransmit $\ell$ via a ring VC. */
      IF ($F_{succ}[a]$ = FALSE) THEN forward $\ell$ via $RVC_{succ}(x)$.
      IF ($F_{pred}[a]$ = FALSE) THEN forward $\ell$ via $RVC_{pred}(x)$.
      Set up a new r_timer($\ell$).
   ENDIF
ENDIF
```

Figure 4.6: The timeout handler in the ER SAF protocol.

a ring within each network segment during partitioning periods, and re-construct a new ring when two or more segments merge into one.

Within each segment, the construction of the virtual ring R comprises three steps. First, the leader switch in the segment, elected by the ATM leader election protocol, computes an ordering of the switches that are reachable from the leader according to the local network image of the leader. Second, the leader switch advertises the ordering using the Basic SAF protocol; such an LSA is called a *switch ordering LSA*. Third, every switch establishes a ring VC to its successor as defined by the ordering. If we define the criterion of the ordering computation to be the minimum total lengths of RVCs, then the switch ordering problem becomes the well-known traveling salesman problem [56]. Since the problem is NP-complete, we use the following heuristic [56]: the leader computes a depth-first search tree that spans all reachable switches in its local network image, and uses the ordering determined by the pre-order traversal of the tree.

When a switch x receives a switch ordering LSA, it sets up a VC to the succeeding switch defined by the ordering, following the procedures described in ATM UNI 3.1 standard [30]. It also accepts a VC-setup request from switch y, where y is the predecessor of x in that ordering. The paths of the two ring VCs are recorded, so that, if subsequent link-down LSAs are received, switch x can detect damage to incident ring VCs.

The maintenance phase of the virtual ring is divided into two levels: repair and reconstruction. When a switch x learns of damage to its $\mathrm{RVC}_{\mathrm{succ}}(x)$ from a link-down LSA, it shall try to establish a new VC to $\mathrm{Succ}(x)$. If this task succeeds, the ring is repaired and no further action is needed. Otherwise, the network has been partitioned. Under such circumstances, a leader will be elected within each segment, and will subsequently compute and flood an ordering of switches within the segment. Switches within each segment then follow the new ordering to establish new ring VCs, resulting in a new ring within the segment.

Another situation requiring ring reconstruction occurs when network segments re-unify with each other (possibly because malfunctioning network components have recovered). As in the previous case, a leader election will take place, and the new leader will compute and flood a switch ordering of the merged segment. In general, the leader of a network/segment monitors the set of reachable switches defined by its local network image. When a membership change occurs to this "reachable set," or when the leader is newly elected, a (new) ordering of switches in the set is computed and flooded, resulting in the (re)construction of the virtual ring.

4.5 Performance Evaluation

We studied the performance of the ER SAF protocol through simulation. The simulator is based on the CSIM package [48]. Confidence intervals were computed, but for most cases are very small and, for clarity, are not shown in the plots.

Networks comprising up to 400 switches were simulated. Since each switch is likely to be attached to several hosts, such networks may include thousands of hosts. For each graph, 40 graphs were generated randomly, and 100 simulation runs were performed on each graph. Each run used a randomly selected core node for SMC construction and a randomly selected flooding source. The core node selected for a simulation session is also used as the root of the depth-first search tree, which determines the ring topology. Table 4.2 shows the characteristics of the graphs generated. These random graphs exhibit average node degrees conforming to those observed in some subnetworks in the Internet [57].

	degree			diameter		
size	min.	avg.	max.	min.	avg.	max.
10	1.95	3.87	5.80	2	2.950	4
20	1.25	3.86	7.28	3	4.475	6
40	1.05	4.02	7.95	5	5.700	8
60	1.02	3.92	8.65	5	6.475	9
80	1.00	4.00	9.38	6	7.000	8
100	1.00	4.10	9.45	6	6.975	8
120	1.00	4.18	10.22	6	7.450	9
140	1.00	4.23	10.60	6	7.475	10
160	1.00	4.30	10.20	7	7.525	9
180	1.00	4.39	10.68	6	7.625	9
200	1.00	4.43	11.18	7	7.750	9
250	1.00	4.57	11.38	7	7.700	9
300	1.00	4.72	12.05	7	7.950	10
350	1.00	4.88	12.12	7	7.775	9
400	1.00	5.08	12.65	7	7.625	9

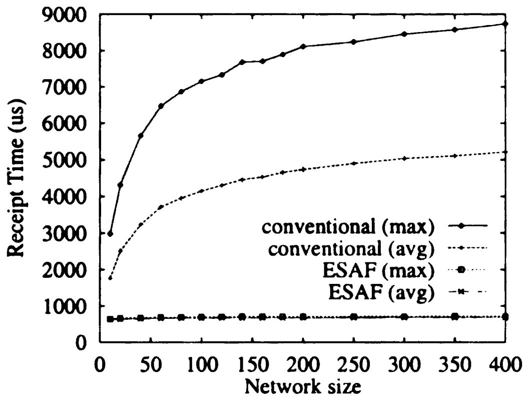
Table 4.2: Characteristics of randomly generated graphs.

Each message transmission in a flooding operation incurs ATM protocol overhead. Like the simulation studies in the previous chapter, we measured these overheads 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-200 switches. From these measurements, we obtained the figure 600 μsec, which includes the overhead at both the sending and receiving switches. The per-hop hardware switching delay was found to be 12 μsec.

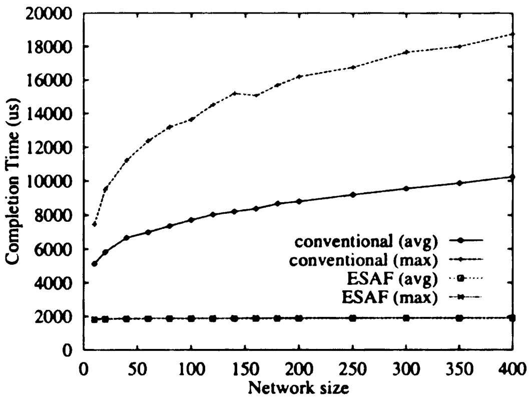
Experiment 1: Normal cases. By normal, we refer to situations where network topology is stable, and the SMC and the virtual ring R are operational. Such circumstances typically occur for the flooding of utilization status information and for periodic flooding. The simulation results pertaining to this setting are plotted in Figure 4.7. For the two time metrics, receipt time and completion time, we show both the average and worst-case results. As we can see in Figure 4.7(a), the ER SAF protocol delivers LSAs several times faster than does the conventional flooding protocol. This is especially true for large networks. When the network size is larger than or equal to 100, the average receipt time of the ER SAF protocol is less than one-fifth of that of the conventional flooding, and the worst-case time is less than one-eighth of that of the conventional flooding. Similarly, the completion time and LSA bandwidth consumption of the ER SAF protocol are only a small fraction of their counterparts in the conventional flooding (see Figure 4.7(b) and (c) respectively). Furthermore, the constant time complexities of the ER SAF protocol are clearly demonstrated by the flat curves.

In Figure 4.7(d), we plot the results regarding the reliability mechanisms of the two flooding protocols. For the conventional flooding protocol, we computed the average number of acknowledgments produced by the 4000 flooding operations for each graph size. For the ER SAF protocol, we compute both the average number of r_acks, as well as the total number of links traversed by these r_acks. The latter metric is of interest in the ER SAF protocol because some ring VCs may involve more than one physical links. As shown in the figure, the average number of r_acks is less than that of the acknowledgments produced by the conventional flooding protocol, although the difference is not as dramatic as the cases in previous figures. We also note that the number of links traversed by r_acks is typically within 150% of the number of r_acks, suggesting that the average length of ring VCs is less then 1.5.

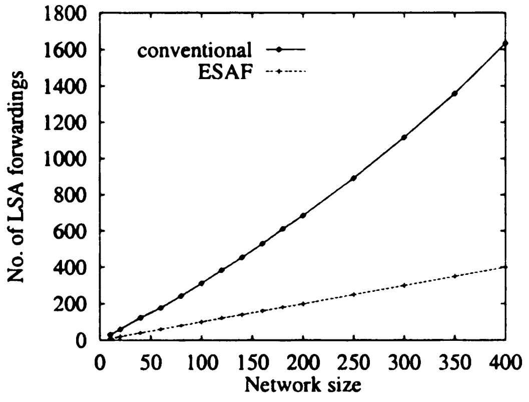
Experiment 2: Flooding of link-down events. Next, we studied the performance of the ER SAF protocol when used to disseminate network component failures, namely link-down events. For each graph, we randomly select and remove a link



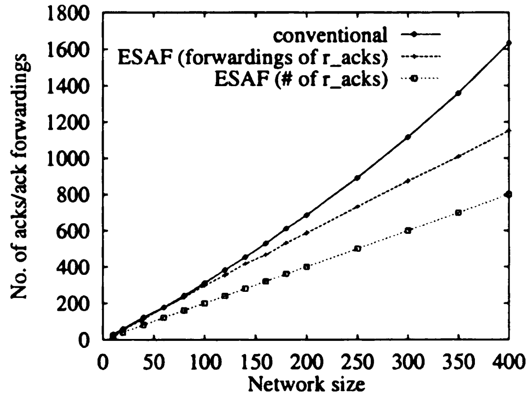
(a) receipt time (μsec)



(b) completion time (μsec)



(c) LSA delivery bandwidth



(d) reliability bandwidth

Figure 4.7: Comparisons of flooding alternatives with an operational SMC and virtual ring.

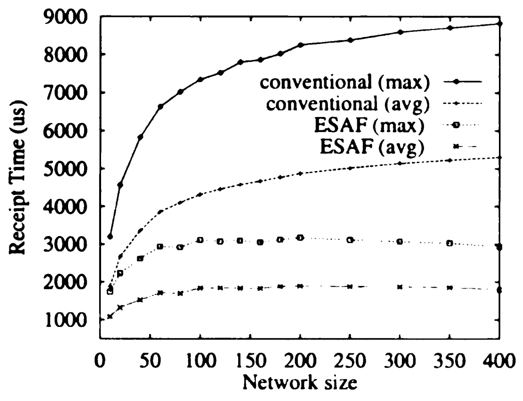
whose removal will not disconnect the graph, and select one of the endpoints of the link to advertise the event. The remaining parts of the network are assumed to be stable. Under the assumption that network components have a long MTBF (mean time between failures), it is unlikely that a second component will fail during the flooding of the first failure (though of course the ER SAF protocol can handle this situation). We expect flooding performance in single-failure scenarios to be representative for the flooding of topology status LSAs.

Corresponding simulation results are plotted in Figure 4.8. As we can see in Figure 4.8(a) and (b), the ER SAF protocol is still much faster than conventional flooding. In most cases of the receipt times, the ER SAF protocol is more than twice as fast as its competitor. We note that a link-down event does not necessarily force the ER SAF protocol to use the G mode. In our simulation, approximately 60% of the link-down events result in the use of the G mode. The performance of the remaining 40% should resemble that of the normal cases. To investigate the performance of the ER SAF protocol in the G mode, we extracted those samples that use mode G and plotted the respective results in Figure 4.8(c). As shown, the average receipt times of LSAs flooded in mode G are only slightly higher than the overall average. This is because the ER SAF protocol also uses the (possibly fragmented) SMC to disseminate LSAs in mode G. After the SMC broadcast, all nodes that receive the LSA start forwarding the LSA via point-to-point links at nearly the same time, resulting in speedier point-to-point forwarding when compared to the conventional flooding protocol. With the presence of G-mode operations, the bandwidth consumptions by the two flooding alternatives are approximately the same, as shown in Figure 4.8(d).

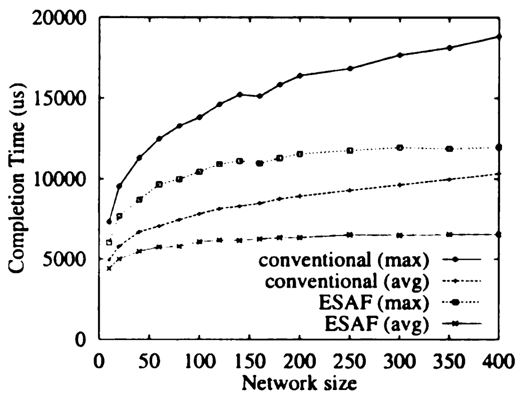
Experiment 3: Ring construction. Lastly, we investigated the time to construct a ring. As discussed earlier, the construction process comprises three phases: First, the leader switch computes switch ordering using the depth-first-search tree heuristic. Second, the leader switch broadcasts the ordering using the conventional flooding protocol. Third, every switch establishes a ring VC to its successor defined in the ordering. Since the first phase is a simply linear time algorithm performed locally at the leader, we ignored this phase in the simulation. The average and worst case ring construction times under this assumption are plotted in Figure 4.9. As we can see, the virtual ring R can be constructed within 22 milliseconds even for relatively large networks.

4.6 Summary

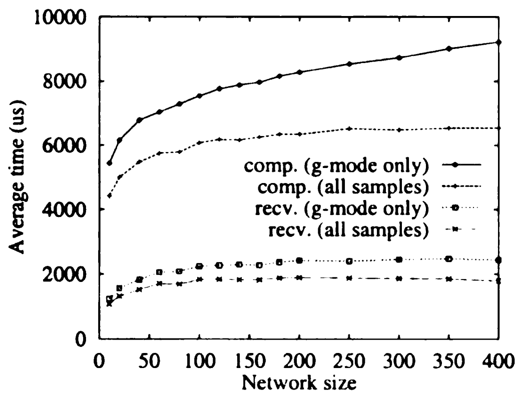
We have described an efficient reliable (ER) SAF protocol. The ER SAF protocol constructs an SMC to broadcast network status updates in hardware and uses a



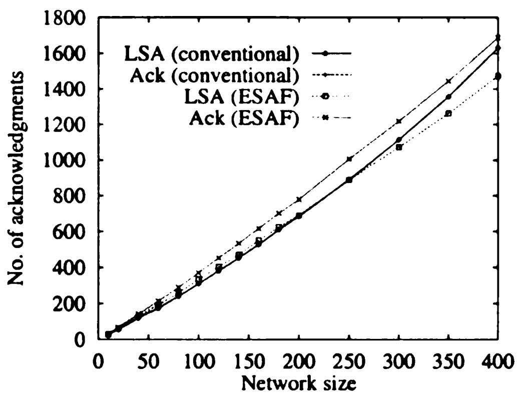
(a) receipt time (μsec)



(b) completion time (μsec)



(c) ER SAF receipt time in different modes (μsec)



(d) bandwidth (the number of links traversed)

Figure 4.8: Comparisons of flooding alternatives in the performance of flooding link-down events.

virtual ring topology to minimize reliability overhead. In normal cases, where the network topology is stable and the SMC and the virtual ring are operational, this protocol is optimal in terms of flooding time and bandwidth consumption. When network component failures affect the operation of the SMC or the ring, the ER SAF protocol resorts to a more conservative flooding method, namely the basic SAF protocol. Our simulation results reveal that the ER SAF protocol is several times faster than the conventional flooding protocol in normal cases, and is still twice as fast as the conventional flooding under adverse circumstances. The use of a ring topology in the ER SAF protocol serves as yet another example of how traditional

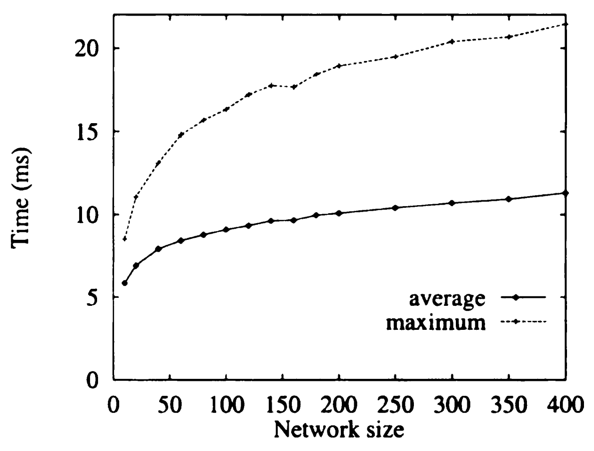


Figure 4.9: The average/worst case time to build a virtual ring.

group communication techniques can be used to improve the performance of LSR.

Chapter 5

A Generic Method of MC Construction

In previous two chapters, we used group communication techniques, namely, the construction of a multiparty communication channel, to improve the performance of LSR in ATM networks. In this chapter, we shift our attention to the other direction of the mutually beneficial relationship between group communication and LSR, that is, how the robustness of LSR and the complete topology information made available by LSR can help develop novel and efficient group communication solutions for use by multiparty communication applications. Specifically, we propose a protocol for the construction and maintenance of multipoint connections (MCs). A distinguishing feature of the protocol is its generality: the proposed solution can incorporate any MC topology computation algorithm, and hence can be used with MCs of different topology types or performance criteria, a requirement stemming from the diversity of multiparty communication applications. The protocol is based on LSR: information regarding multipoint connections is broadcast to network switches, which perform all MC topology computations locally. The protocol is free from routing loops, even transient ones, and will tolerate any combination of link/node failures, including those that partition the network for a period of time. The correctness of the protocol, which is modeled as a consensus problem in a distributed system, is established by formal proofs. Results of a simulation study show that the generality of the protocol can be achieved with negligible to moderate signaling overhead.

5.1 Motivation

As described in Chapter 2, the applications that use multi-party communication vary widely and include teleconferencing, computer-supported cooperative work, distributed interactive simulation, remote teaching, tele-gaming, replicated file servers, parallel database search, and distributed parallel processing. Such applications have widely disparate needs with respect to network services. Among such services is the MC protocol itself, which defines the set of rules and conventions by which MCs are constructed and maintained, and which is executed among processing entities within a communications network.

We have discussed in Chapter 2 three major MC topology types, namely, SSTs, SRTs, and ROSTs. Even for a fixed topology type, different topology computation algorithms could be used, depending on the relative importance of various performance criteria, such as bounds on transmission delays, network resource consumption, multicast packet loss rate, and so forth. Many existing multicast protocols can be considered as distributed implementations of one, or a small set of, MC topology computation algorithms. For example, the MOSPF and DVMRP protocols implement distributed source-rooted tree algorithms that minimize transmission delays, whereas the CBT protocol implements a particular shared-tree algorithm first described in [8]. However, the emerging demand for routing based on quality-of-service (QoS) [58] has stimulated the development of many other MC topology computation algorithms that may be better suited for certain classes of applications [10, 34, 35, 59, 60]. Many such algorithms have not been incorporated into current multicast protocols.

The diversity of MC topology types demanded by multiparty communication applications, and the wide variety of MC topology computation algorithms designed for different performance criteria, give rise to the following question: Is it possible to develop an MC protocol "chassis," that is, a framework that is able to accommodate multiple existing, and future, MC topology algorithms? The ongoing development of new service models (available bit rates, controlled load, quality-of-service, and so on) further emphasizes the need for such a *generic* MC protocol. The main contribution of this chapter is to demonstrate that such a challenging goal is achievable in networks based on LSR.

In this chapter, we propose an LSR-based MC protocol, called the generic MC (GMC) protocol, which can be used as a distributed implementation of "any" MC topology algorithm. The GMC protocol extends LSR by including information about MCs in the network images maintained at switches. MC topology and membership information are broadcast throughout the network by means of extended LSAs. We emphasize that the GMC protocol is intended to construct and maintain MCs among switches, rather than hosts. As discussed in Section 2.2.2, a host in a network is attached to one or more switches, called the *ingress* switch of the host, and uses a local membership management protocol, such as IGMP [7], to inform its respective ingress switch of MC membership status. When one or more attached hosts of a switch are interested in an MC, the switch is said to be a member switch of the MC. Figure 5.1 shows the same MC as depicted in Figure 2.1 (b), complete with member hosts.

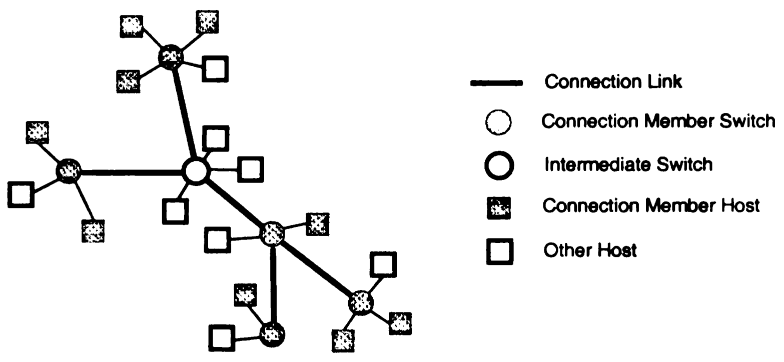


Figure 5.1: Example MC showing member switches and attached hosts.

The primary task of the GMC protocol is to keep MC images consistent and up-to-date, while incurring minimum protocol overhead. Since both network topology information and MC image information are available at all switches, any method of computing MC topologies can be used. Indeed, the topology computation algorithm is a "plug-in" component of GMC, rather than an inherent part of the protocol. In addition to its ability to support multiple MC types and topology algorithms, the GMC protocol exhibits the following properties:

1. *The protocol is free from routing loops.* Due to delays in the dissemination of changes in network status, the participating switches in an MC protocol may have inconsistent knowledge of the network for short periods of time. Many multicast protocols produce transient routing loops under such circumstances [5, 6, 4, 2]. Routing loops, even temporary ones, may introduce network congestion under conditions of heavy traffic. As we will demonstrate later, the GMC protocol avoids routing loop entirely.

2. *The protocol is robust.* Being a link-state routing protocol, the GMC protocol has the intrinsic advantage of fault tolerance. The protocol handles faulty components in the network through topology computations that are triggered by link/nodal events. In fact, the protocol survives network partitioning, and is able to construct correct MC topologies after re-unification. Further, we will show that the GMC protocol can survive memory overflow problems at switches: given an MC, the protocol will be able to construct the MC as long as at least one member of the MC does not purge the image of the MC indefinitely.

3. *The protocol exhibits a low level of computational redundancy compared to existing LSR-based MC solutions.* As we will discuss in Section 5.2, LSR-based MC solutions, such as the MOSPF protocol, may perform identical topology computations at multiple switches, incurring the problem of computational redundancy. Since an MC topology computation is typically a non-trivial task (for example, many of the Steiner tree heuristics are of $O(N^2)$ or $O(N^3)$ complexities), the GMC protocol is designed to minimize the number of topology computations. We will show via a simulation study that the performance of GMC in terms of computational overhead compares favorably with that of MOSPF.

Given the availability of network status information at all network switches, LSR provides a solid foundation for developing an MC protocol chassis such as GMC.

While the idea of LSR-based MC protocols is not new, previous solutions have not achieved the versatility that is dictated by the diversity of multiparty communication applications. This paper is a "proof of concept" that, at least in LSR-based networks, such a generic MC protocol can be constructed and can operate efficiently.

The remainder of this chapter is organized as follows. Section 5.2 discusses various background subjects as well as related work. Section 5.3 describes the design and operation of the GMC protocol. We prove in Section 5.4 the correctness of the GMC protocol. Section 5.5 presents the results of a simulation study, in which the behavior of the GMC protocol is evaluated under various workloads. A summary of this chapter is given in Section 5.6.

5.2 LSR-Based Multipoint Connections

As described in Chapter 2, switching elements in LSR-based networks use LSAs to advertise local status information. This approach can be extended to support multiparty communication by distributing MC membership information in LSAs [1]. That is, whenever a switch wants to join or leave an MC, a membership-event advertisement for the connection is flooded through the network. Such an advertisement should contain at least the ID of the connection and the address of the source switch. Switches in the network collect these advertisements and maintain member lists and MC topology information for all active MCs. The differences among LSR-based MC protocols lie in how topology computations are triggered.

The MOSPF protocol is an extension of the unicast OSPF protocol [11]. In the MOSPF protocol, the addresses of the hosts listening to a multicast address are broadcast in group-membership LSAs, and routers maintain complete member lists for all active multicast addresses. However, a router does not compute the topology of a multicast connection until it actually receives a datagram destined for the corresponding multicast address. Upon receiving a datagram for multicast address M, the router consults its local database for the member list of M and computes a shortest-path tree, rooted at the source of the datagram, that reaches all hosts listening to M. The router then saves this topology information in a routing cache and forwards the datagram along the appropriate outgoing links. This forwarding will trigger further topology computations at downstream routers. Moreover, multicast routing entries created in this process must be cleared upon the arrival of LSAs that advertise membership or network changes, resulting in the repetition of the process when subsequent multicast datagrams arrive.

The MOSPF approach (*on-demand*, *data-driven* topology computations) is well-suited to the construction of source-rooted trees. However, this method has limitations in other contexts. First, if the MC is an ROST, it is independent of the nodes sending to that MC; its topology computations cannot be triggered by packets from senders, but rather depend on the actions of receivers. Second, the MOSPF performs identical topology computations at all members and intermediate nodes of an MC. This computational redundancy could produce heavy workloads at switches, given the high cost of topology computations. (MOSPF uses Dijkstra's shortest path algorithm, which exhibits complexity $O(N^2)$, where N is the number of switches in a network.) Third, the MOSPF protocol requires the availability of MC membership information at a router to compute the topology of an MC. Losses of such information (for example, due to memory overflows) could lead to improper protocol operation. To summarize, while the MOSPF protocol serves some multicast scenarios well, it may not possess the efficiency and flexibility to accommodate many current and future distributed applications.

5.3 The GMC Protocol

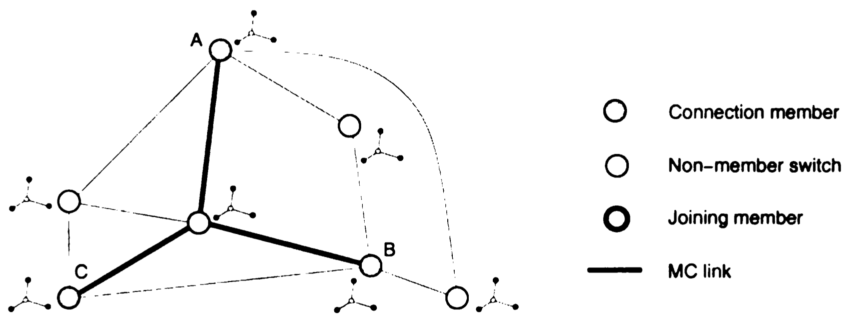
The GMC protocol extends LSR by incorporating MC images at network switches. The essence of the protocol is to maintain consistency of MC images throughout the network. In presence of group membership dynamics or changes to network topology, an MC protocol must update the MC topologies that are affected by those events. The GMC protocol uses an event-driven approach to this problem: the switches that detect events are required to compute and advertise new MC topologies. For example, a switch that detects a "link-down" event suggests alternative topologies for any MCs that were using the malfunctioning link. Similarly, a switch that changes its membership status with respect to an MC implicitly "detects" a membership change event, and suggests a new topology for that MC. Other switches in the network receive the topology proposal and, if it is accepted, modify MC links and/or routing entries accordingly.

5.3.1 Design Issues

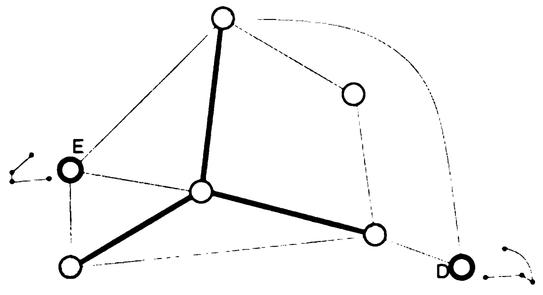
A major problem that the GMC protocol must solve is the proposal of multiple, inconsistent MC topologies by switches that detect different events at nearly the same time. An example of this problem is illustrated in Figure 5.2. The example begins with the network and MC configuration that are depicted in Figure 5.2(a), where nodes A, B, and C are members of an MC. Let us assume that switches D and E request to join the MC at approximately the same time. Without knowledge of each other's intentions, switch D sees member list (A, B, C, D) and proposes a topology spanning those four nodes, while switch E sees member list (A, B, C, E) and proposes a different topology (see Figure 5.2(b)). If updates to routing table entries are not handled properly, the two inconsistent proposals could result in a routing loop shown in Figure 5.2(c).

In the GMC protocol, the inconsistent-proposal problem can be detected by embedding membership information in MC topology proposals. In the above scenario, for example, if node D notices that itself is absent from E's topology proposal, and node E notices a similar flaw in the proposal from D, then both of them will subsequently compute new (and correct) proposals. We will show in the formal presentation of GMC that, while this example concerns membership inconsistency problems, the same method is used to cope with inconsistency problems created by simultaneous network topology status changes.

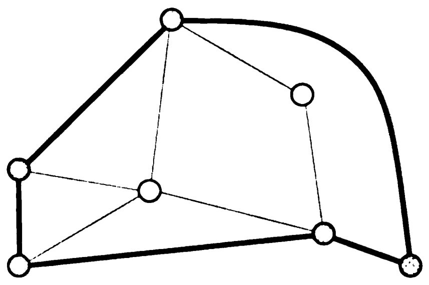
During a busy period when multiple events take place concurrently, multiple proposals may be suggested and flooded through the network. Although some of these proposals are more up-to-date than others, the underlying flooding mechanism has



(a) original MC containing A, B, C.



(b) switches D and E request to join, and propose inconsistent topologies.

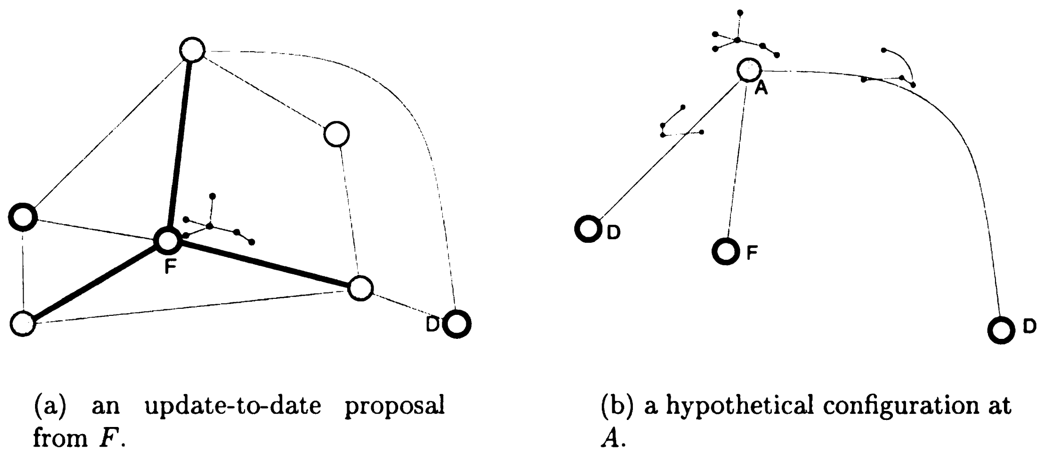


(c) potential routing loop.

Figure 5.2: Problem created by inconsistent topology proposals.

no such knowledge and may deliver proposals in any order. Consider the example shown in Figure 5.3, which continues the scenario in Figure 5.2. Here, we assume that switch F also requests to join the MC, after receiving the proposals from D and E. Figure 5.3(a) depicts the MC topology suggested by F. As shown, this proposal contains update-to-date membership information and should override proposals from other switches. It is possible, however, for the switch A to receive F's proposal before receiving the earlier ones (perhaps because A ignored proposal advertisements from D and E due to the lack of buffer space but later recovers and is able to accept the proposal from F). The proposals from D and E will eventually arrive at A by means of retransmission, incorrectly overriding the up-to-date MC image already established at A (see Figure 5.3(b)). The GMC protocol uses the well-known timestamp technique [61] to resolve this proposal ordering issue.

Another desirable property of MC protocols is freedom from temporary routing



(a) an update-to-date proposal from F.

(b) a hypothetical configuration at A.

Figure 5.3: The topology ordering problem. If F's proposal is received before those of D and E, these obsolete proposals will override the update-to-date MC image at A.

loops. In the previous example, even if inconsistent proposals are detected and eventually resolved, any routing loop, however transient, can quickly leading to traffic congestion if heavy traffic loads are placed on the MC during that period. In the GMC protocol, a topology proposal is uniquely identified by its source switch ID and its timestamp value. This stamp-source pair serves as the ID of an MC topology. To prevent loops, the two switches at the ends of an MC link exchange the IDs of their local MC images before establishing the link as part of the MC. Only if the two IDs identical will the MC link be established. Using this check, MC links, such as those in the loop of Figure 5.2(c), cannot all be permitted, because somewhere in the loop two adjacent nodes must have different MC topologies, and hence, different topology IDs.

5.3.2 Protocol Overview

With the above design issues in mind, the operation of the GMC protocol can be summarized as follows.

- Every switch x maintains a timestamp $R_{x,m}$ for every active MC m. The value of this timestamp is set to the largest timestamp value among the received LSAs relating to m. The switch will ignore topology proposals about the MC m with stamp values less than or equal to $R_{x,m}$.

- Every switch x maintains a *mailbox* for every active MC. The mailbox stores received, but not yet processed, LSAs that are relevant to the MC.

- When the switch detects a local event that affects the MC m (for example, the switch changes its membership status respect to m or detects failure of an incident link that is used by m), the switch creates and floods an *event* LSA, which describes the event. The LSA may also contain a new topology proposal if the mailbox for m is empty. (There is no reason to compute a new topology for m if information regarding m from other switches is yet to be processed.)

- When the switch receives a topology proposal P for m, it checks the proposal for consistency problems. The receiving switch checks only "local" inconsistencies, that is, it checks if its own membership status in P conforms with its current membership status and if P includes any malfunctioning incident links of the switch. If a local inconsistency is detected, the switch *objects* — it computes and advertises a new topology proposal. LSAs that carry proposals produced in this manner are called *triggered* LSAs.

- A topology proposal P is *accepted* at a switch x if the switch finds no local inconsistency for P and if the timestamp of P is greater than $R_{x,m}$.

- To prevent the GMC protocol from being overly reactive to bursts of events, topology computations are subject to a *hold-down* period. The hold-down period guarantees that successive topology computations must be at least Δt seconds apart. Assuming that the current image for the MC m at switch x is received or computed at time α, and that the switch is ready to compute a new topology at time β where $\beta - \alpha < \Delta t$, the switch sets up a timer, called `TC-TIMER` (TC stands for Topology Computation), with length $\Delta t - \beta + \alpha$. The postponed topology computation is resumed if no locally consistent topology proposals are received before the timer fires. The proper choice of the Δt value is a subject of our performance study in Section 5.5.

5.3.3 GMC LSA Format

Before we present the details of the GMC protocol, we must define the format of LSAs. We use the term *non-GMC LSA* to refer to an advertisement produced and processed by the underlying unicast LSR protocol, and the term *GMC LSA* to refer to an advertisement produced by the GMC protocol.

In a network comprising n switches, a non-GMC LSA is a tuple (S, seq, F, D), where S is the source of the LSA, seq is the sequence of the LSA, F with value $\overline{gmc}$ indicates that the LSA is used for the advertisement of a link/nodal event, and D encodes a description of the event. For example, a description of a link-down event must include at least the two end switches of the link. The exact format of link/nodal event descriptions is defined by the underlying unicast LSR protocol, and is not discussed further.

A GMC LSA is a tuple (S, seq, F, V, G, P, T), where $S \in \{0, 1, \ldots, n-1\}$ is the source address of the LSA, seq is the sequence number of the LSA, F with value gmc identifies this LSA as an GMC LSA, $V \in$ {join, leave, link, none} specifies an event from the source switch S, G identifies the MC to which this LSA is relevant, P is either a topology proposal for G or the member list of G, and T is a timestamp. An event of type "link" in a GMC LSA indicates that a link/nodal event affects the topology of an MC. Specifically, a link/nodal event will cause the unicast LSR protocol to produce exactly one non-GMC LSA and will cause the GMC protocol to produce k GMC LSAs, where k is the number of MCs whose topologies are affected by the event. We use the configuration shown in Figure 5.4 to illustrate. Let us assume that the following events occur: switch X intends to join connection C_1, switch E wishes to leave connection C_2, and switch F detects the failure of the (F, B) link. As shown in Figure 5.5, the three events trigger five advertisements: one for the join event, one for the leave event, and three for the link event.

Given a link/nodal event that occurs at switch F, we assume that switch F floods the single corresponding non-GMC LSA before flooding the corresponding GMC LSAs. We further assume that the sequence number of a non-GMC LSA will be smaller than those of the corresponding GMC LSAs. A switch that receives an LSA out of order will not process it until the switch has received preceding LSAs. Therefore, at any receiving switch, the processing of the non-GMC LSAs (by the unicast LSR protocol) will precede the processing of the k GMC LSAs (by the GMC protocol). Hence, the event advertised in the non-GMC LSA will have been incorporated in the local network image at the switch before the ensuing GMC LSAs are used to update MC images.

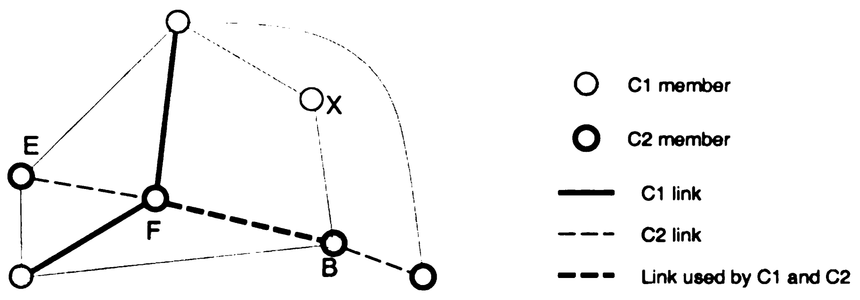


Figure 5.4: A network/MC configuration.

As demonstrated in the previous example, the GMC protocol produces a set of GMC LSAs that disseminate all events relevant to an MC. (For example, in Figure 5.5, the protocol produces two GMC LSAs for connection C_1: one for the join of X and another for the failure of the (F, B) link.) Therefore, the algorithms of the GMC protocol can be presented in a per-MC manner without loss of generality.

5.3.4 Data Structures And Protocol States

Besides the aforementioned timestamp $R_{x,m}$, every switch x in the network maintains a variable *last_tc_time*$_{x,m}$ (last topology computation time) for each MC m, and uses a $mailbox_{x,m}$ to store incoming GMC LSAs regarding m. Every switch x in the network also maintains a local image for each MC m, denoted by *Image*$[x, m]$. An MC image includes the topology of the MC (denoted by $P($*Image*$[x, m]))$, the ID of the switch that proposed the topology (denoted by $S($*Image*$[x, m]))$, and the timestamp of the topology (denoted by $T($*Image*$[x, m]))$. The switch x further maintains a list of MC members, denoted by *Members*$[x, m]$, and a real time clock, *clock*$[x]$. In the following

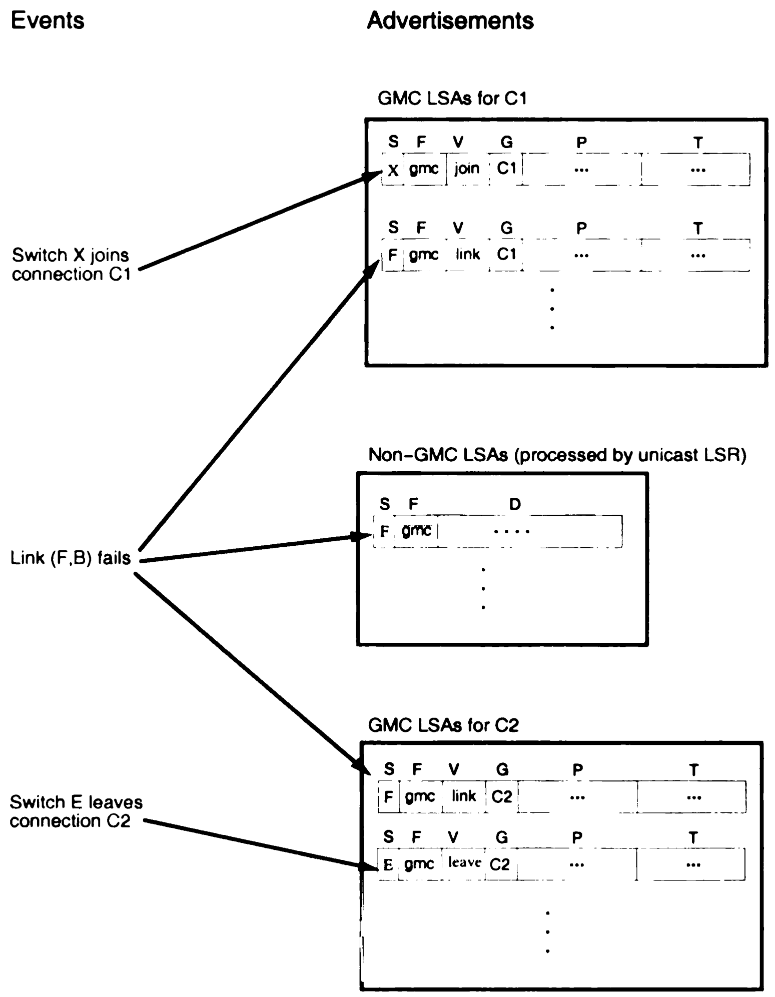


Figure 5.5: Events and advertisements in the GMC protocol.

discussion, subscripts and indices in this notation may be omitted, if they are clear from the context.

With respect to an MC m, the GMC protocol at a switch can be in one of the four states shown in Figure 5.6: EVENT-HANDLING, RECEIVING-LSA, DELAYED-TC, or IDLE. Initially, the GMC protocol is in the IDLE state. Whenever an event relating to m occurs at a switch, the switch moves into the EVENT-HANDLING state and invokes its **EventHandler** routine. This routine creates and floods an event LSA, which describes the event and may also contain a new topology proposal. Whenever GMC LSAs are present in the mailbox of m at a switch, the switch enters the RECEIVING-LSA state, invokes the **ReceiveLSA** routine to process the incoming LSAs, and checks for inconsistency problems before accepting the topology pro-

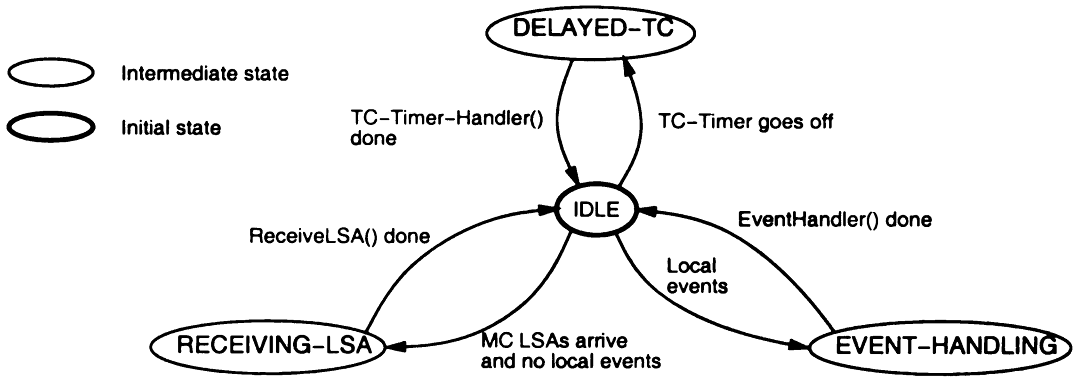


Figure 5.6: The state-transition diagram of the GMC protocol.

posal, if present, in the LSA. After the completion of either the **EventHandler** or the **ReceiveLSA** routine, the switch returns to the IDLE state. When a hold-down timer fires, the switch enters the DELAYED-TC (Delayed Topology Computation) state and invokes the **TCTimerHandler** routing. When local events, LSA arrivals, and timer firings occur simultaneously, the EVENT-HANDLING state has priority, followed by the RECEIVING-LSA state.

5.3.5 Protocol Algorithms

We are now ready to describe the algorithms for **EventHandler**, **ReceiveLSA**, and **TCTimerHandler**. In the following, we assume that the floodings of LSAs are reliable and that LSAs from the same switch are ordered by sequence number. Reliable flooding can be implemented using either a reliable hop-by-hop protocol, or by periodic re-flooding [12]. Different reliability mechanisms and flooding algorithms affect the timing behavior of the GMC protocol, but do not affect its correctness.

At a switch x and given an MC m, GMC algorithms share the data structures described in the previous section. (Additional variables shared by these algorithms, such as the *make_proposal_flag* variable, will be introduced later.) We point out that simultaneous accesses to shared data structures and variables cannot occur, because at any moment of time the GMC protocol is in exactly one of the four states shown in Figure 5.6, and will leave the current state only if the corresponding routine is completed.

The **EventHandler** algorithm is given in Figure 5.7. The algorithm is presented in a per-MC manner, that is, when an event occurs, this routine is invoked for every connection affected by the event. This protocol entity is responsible for the generation of GMC LSAs only; the non-GMC LSA resulting from link/nodal event is generated and flooded by the underlying unicast protocol. In Figure 5.7, the local switch is identified by parameter x, the event is given in parameter ev, and the affected connection is given by parameter m. The **EventHandler** may be invoked because of membership change events (that is, when switch x joins or leaves the MC m), or link state events that affect the MC (for example, an incident link that is used by the MC fails). In both cases, the routine advances the timestamp R of m (line 1), updates MC member list when necessary (lines 2-4), and computes a new MC topology P for m (line 6), if such an action is not prohibited by a hold-down period. If a new topology is not computed due to the hold-down period, then the **TC-TIMER** is set up at line 9 to defer the computation to **TCTimerHandler** (if the timer is already in use, line 9 restarts the timer). Even if the computation at line 6 is performed, the result P may be obsolete after the completion of the computation, due to the arrivals of new GMC LSAs regarding m. If P remains up-to-date after computation, it is flooded throughout the network (line 14) and accepted at x itself (by calling an auxiliary routine, **AcceptTopology**, at line 15). When the proposing of a topology is postponed due to either the hold-down period or obsolescence, the **EventHandler** at line 17 floods the event ev with a member list of m, rather than an MC topology, and defers to the **ReceiveLSA** routine to make sure that a correct MC image is eventually established. This information is passed to **ReceiveLSA** by setting a shared variable, *make_proposal_flag*, equal to TRUE (line 18). As with other GMC variables, at each switch x there is one *make_proposal_flag* variable for each MC m.

The **AcceptTopology** algorithm, shown in Figure 5.8, registers an MC topology P into the local database of the invoking switch x, and attempts to establish incident MC links according to the new topology. The local MC image *Image*, including the topology, the source switch ID, and timestamp, are updated at lines 1 to 3. The routine then tries to establish MC links that are defined in P and incident to x.

```
Algorithm: EventHandler
Input: switch ID $x$, event $ev$, and connection $m$

 1: $R = R + 1$.
 2: IF ($ev$ is for membership status change of $x$)
 3:     Update Members($m$) accordingly.
 4: ENDIF
 5: IF ($clock - last\_tc\_time > tc\_holddown$),
 6:     Compute a new topology proposal $P$ for the connection $m$.
 7:     $last\_tc\_time = clock$.
 8: ELSE
 9:     Set the TC-TIMER to value $tc\_holddown - clock + last\_tc\_time$.
10: ENDIF
11: IF (a new topology $P$ is computed) and
12:     (no LSAs for $m$ received during the computation),
13:     /* proposal is still valid */
14:     Flood the GMC LSA $(x, gmc, ev, m, P, R)$.
15:     AcceptTopology ($x$, $m$, $P$, $R$, $x$).
16: ELSE /* flood event but defer to ReceiveLSA to make proposal */
17:     Flood the GMC LSA $(x, gmc, ev, m, Members(m), R)$.
18:     $make\_proposal\_flag$ = TRUE.
19: ENDIF
```

Figure 5.7: The algorithm for **EventHandler**.

As described earlier, an MC link (x, y) is established only if the MC image at the neighboring switch y has a source switch ID and a timestamp identical to those at x (lines 6-9). Before completion, the routine sets the *make_proposal_flag* variable to FALSE, and records the current time in *last_tc_time*.

The algorithm for the **ReceiveLSA** routine is given in Figure 5.9. Parameter x identifies the local switch, and parameter m specifies the MC. The routine is invoked when the switch enters the RECEIVING-LSA state, that is, when there is at least one LSA in mailbox for connection m. For every such LSA, **ReceiveLSA** updates the local member list of connection m if the event in ℓ is about a membership change (line 3). Next, the routine checks for inconsistency problems in the LSA and records the result in a variable, *my_status_consistent* (lines 4-9). As mentioned earlier, the switch x is only interested in local consistency, that is, the received LSA must contain correct membership information with respect to x (line 5) and any topology proposal in the LSA must not use any malfunctioning links that are incident to x (line 4).

```
Algorithm: AcceptTopology
Input: switch ID $x$, connection $m$, topology $P$, stamp $T$ and source $S$.

 1: $P(Image[x][m]) = P$.
 2: $S(Image[x][m]) = S$.
 3: $T(Image[x][m]) = T$.
 4: FOR(every link $t$ in $P$ that is incident to $x$) DO
 5:     Let $t$ be an $(x, y)$ link.
 6:     Exchange messages with $y$ to learn $S(Image[y][m])$ and T(Image[y][m]).
 7:     IF ($S(Image[x][m]) = S(Image[y][m])$) and ($T(Image[x][m]) = T(Image[y][m])$)
 8:         Establish $(x, y)$ link for connection $m$.
 9:     ENDIF
10: ENDDO
11: $make\_proposal\_flag$ = FALSE.
12: $last\_tc\_time = clock$.
```

Figure 5.8: The algorithm for **AcceptTopology**.

Next, the routine decides if the LSA can be accepted (lines 10-13). For an LSA ℓ to be accepted it must include a topology proposal that is more recent than the local one and that is locally consistent. The LSA ℓ is more up-to-date than the local MC image at x if it is tagged with a larger timestamp value; a tie in the timestamp comparion is resolved by the values of source switch IDs (line 12). If the LSA is accepted, then the **AcceptTopology** routine is invoked to update the local MC image (line 14), and the *make_proposal_flag* for connection m is set to FALSE (line 15), since an up-to-date topology for the connection has been accepted. Otherwise, the switch checks whether its local status is consistent with the received topology proposal (line 17). If not, then the switch plans to construct a new topology proposal by setting its *make_proposal_flag* variable to TRUE (although it may need to process additional LSAs first). To conclude the processing of the current LSA ℓ, the **ReceiveLSA** routine advances the R timestamp for MC m to be at least as large as that of ℓ (line 19). Since the R timestamp will be advanced again before the switch x proposes and floods any topology in the future (line 1 of **EventHandler**, line 31 of **ReceiveLSA**, and line 8 of **TCTimerHandler**), the advancement at line 19 ensures that subsequent topology proposals will be tagged with timestamps larger than anything x has received.

After consuming all the LSAs in the mailbox, the **ReceiveLSA** routine decides

```
Algorithm: ReceiveLSA
Input: switch ID $x$, connection ID $m$.

 1: WHILE (there are LSAs for connection $m$ in mailbox)
 2:     Get next LSA $\ell = (S, gmc, V, m, P, T)$.
 3:     Update member list of $m$ accordingly, if $V$ is for membership update.
 4:     IF (for all link $t$ used in $P$ that is incident to $x$, $t$ is ON) and
 5:        (the membership of $x$ in $P(\ell)$ is consistent with that in Members($m$)),
 6:        $my\_status\_consistent$ = TRUE.
 7:     ELSE
 8:        $my\_status\_consistent$ = FALSE.
 9:     ENDIF
10:     IF ($P(\ell)$ is a topology proposal) and
11:        ($T(\ell) \geq R$) and
12:        ($T(\ell) > T(Image)$, or ($T(\ell) = T(Image)$ and $S(\ell) > S(Image)$) and
13:        ($my\_status\_consistent$= TRUE)),
14:        AcceptTopology ($x$, $m$, $P(\ell)$, $T(\ell)$, $S(\ell)$).
15:        $make\_proposal\_flag$ = FALSE.
16:     ELSE
17:        $make\_proposal\_flag$ = TRUE, if ($my\_status\_consistent$ = FALSE).
18:     ENDIF
19:     $R = \max\{R, T(\ell)\}$.
20: ENDWHILE
21: IF ($make\_proposal\_flag$ = TRUE)
22:    IF ($clock - last\_tc\_time > tc\_holddown$),
23:       Compute a new topology $P$ for the connection $m$.
24:       $last\_tc\_time = clock$.
25:    ELSE
26:       Set up the TC-TIMER with length $tc\_holddown - clock + last\_tc\_time$.
27:    ENDIF
28:    IF (a new topology $P$ is computed) and
29:       (there are no LSAs in mailbox for connection $m$) and
30:       (no local events for connection $m$ queued at $x$),
31:       $R = R + 1$.
32:       Flood ($x, gmc$, none, $m, P, R$).
33:       AcceptTopology ($x$, $m$, $P$, $R$, $x$).
34:    ENDIF
35: ENDIF
```

Figure 5.9: The algorithm for **ReceiveLSA**.

whether a new proposal should be computed, depending on the value of the *make_proposal_flag* variable (line 21) and the hold-down mechanism (line 22). If a topology is computed at line 23, two conditions must be satisfied before the proposal is actually flooded: 1) no new GMC LSAs arrive during the computation period (line

29), and 2) no local events take place during the period (line 30). If the proposal is still up-to-date at the end of computation, then it is flooded to the other switches and accepted locally (lines 32-33). Otherwise, it is withdrawn and the *make_proposal_flag* remains true, indicating the lack of an up-to-date MC image for m at switch x. In the case where the topology computation is held down, the **TC-TIMER** is set up (or restarted, if it is already in use) at line 26 to defer to computation to **TCTimerHandler**.

The algorithm for the **TCTimerHandler** routine is given in Figure 5.10. Again, parameter x identifies the local switch, and parameter m specifies the involved MC. Before resuming a postponed topology computation, the routine first checks if this computation is still needed. The computation may no longer be necessary because, during the hold-down period, topology proposal(s) may have been received and accepted (hence setting the *make_proposal_flag* to FALSE), or there may be pending "news" about the MC (GMC LSAs in the mailbox or events in the event queue). Similar to the previous routines, the new topology P is actually flooded only if no further news about the MC is observed during the computation period.

```
Algorithm: TCTimerHandler
Input: switch ID $x$ and connection $m$.

 1: IF (make_proposal_flag= FALSE) and
 2:     (there are no LSAs in mailbox for connection $m$) and
 3:     (no queued events for connection $m$),
 4:     Compute a new topology $P$ for the connection $m$.
 5:     last_tc_time = clock.
 6:     IF (there are no LSAs in mailbox for connection $m$) and
 7:         (no queued events for connection $m$),
 8:         $R = R + 1$.
 9:         Flood $(x, gmc, \text{none}, m, P, R)$.
10:         AcceptTopology($x$, $m$, $P$, $R$, $x$).
11:         make_proposal_flag = FALSE.
12:     ENDIF
13: ENDDO
```

Figure 5.10: The algorithm for **TCTimerHandler**.

5.3.6 MC Creation and Destruction

The creation and destruction of an MC require no special mechanisms. When the first member of an MC advertises its presence, the other switches allocate necessary data structures for the MC and accept the topology proposal contained in the advertisement. When a switch detects an empty member list of an MC, local data structures corresponding to the MC are deleted.

5.4 Proof of Correctness

In this chapter, we formally show the correctness of the GMC protocol in two steps. In the first step, we consider the correctness of the protocol without memory shortage problems (that is, operational switches will not lose MC images). Under this assumption, we will show that, given a finite set of events, the algorithm will reach consensus about MC images among network switches by producing a finite number of LSA broadcasts. (Our simulation results, presented in Section 5.5, show that in practice the number of LSAs per event is likely to be small.) In the second step, we describe (minor) extensions to the GMC protocol to handle losses of GMC data structures, and establish a sufficient condition for the GMC protocol to work correctly in presence of such switch memory overflows. For clarity, the discussion in this section is in terms of a single MC. As illustrated in Figure 5.5, the GMC protocol, when given a set of events Π (link-state and/or MC membership changes), produces a set of GMC LSAs, $\mathcal{L}_m$, exclusively for every MC m. Thus, the protocol activities associated with different MCs proceed independently; herein lies the generality of a proof regarding a single MC.

5.4.1 Correctness without Memory Overflows

Under the assumption that switches, unless crashed, will not lose MC images, the GMC protocol proceeds as described in Section 5.3. In the following discussion, we assume a finite set of events, denoted as Π, that does not leave the network permanently partitioned. Our goal is to show that such an even set will not lead to infinitely looping GMC activities. We point out that temporary partitioning could be produced by such an event set Π; all but permanent partitions are handled by GMC.

Lemma 1 *Given an MC m and a set of events Π as defined above, the GMC protocol produces a finite set of LSAs, $\mathcal{L}_m$.*

Proof: Since flooding operations are assumed to be reliable (see Section 5.3), there exists a time τ by which all the events in Π are learned by all switches. (If Π incurs temporary partitioning, reliability of flooding can be enforced by periodic re-flooding.) Any GMC LSA produced after time τ will incorporate the changes in Π. Such an LSA will not be objected to by any other switch (that is, the corresponding *my_status_consistent* values will be TRUE at all switches), and hence will not trigger any additional LSAs. Since an LSA must require a minimum time Δt to construct, we see that the GMC protocol is able to produce only a finite number of LSAs by time τ, and hence the set $\mathcal{L}_m$ must be finite. $\square$

In the following discussion, a GMC LSA is said to be (locally) consistent at a switch y if the ensuing *my_status_consistent* value is TRUE at y. Also, we will drop the subscript m in the notation $\mathcal{L}_m$, since all the discussions are about an MC m that has at least one member join event in Π (otherwise, the MC is inactive and is not relevant).

Definition 1 *We denote by ℓ_{max} the LSA in $\mathcal{L}$ that has the maximum timestamp-source pair $(T(\ell_{max}), S(\ell_{max}))$.*

The concept of the maximum element in $\mathcal{L}$ is well defined because the set $\mathcal{L}$ cannot be empty; at least one GMC LSA is generated for the member join assumed above. We will see that all network switches will accept $\ell_{\max}$, and the topology contained in $\ell_{\max}$ will be the consensus among all network switches, due to the two properties stated in subsequent lemmas. Recall that $S(\ell)$ and $T(\ell)$ are the source and timestamp of an LSA ℓ.

Lemma 2 *The LSA ℓ_{max} includes a topology proposal.*

Proof: Let us assume the opposite. A GMC LSA that does not contain a topology proposal must be produced by the EventHandler routine at line 17, a scenario that occurs when there are incoming GMC LSAs during the topology computation of ℓ_{max}. If this happens to ℓ_{max}, the value of R at switch $S(\ell_{\text{max}})$ at this moment is $T(\ell_{\text{max}})$, and the *make_proposal_flag* variable is set to TRUE.

Consider the GMC protocol activities at $S(\ell_{\text{max}})$ after the production of ℓ_{max} (a GMC activity is an invocation of EventHandler, ReceiveLSA, or TCTimerHandler). The ReceiveLSA must be invoked at least once, to process the LSA(s) that arrived during the processing of ℓ_{max}. We exclude the possibility of post-ℓ_{max} events occurring at $S(\ell_{\text{max}})$; otherwise, further invocations of EventHandler will advance the timestamp R, and flood LSAs with timestamps greater than that of ℓ_{max}, a contradiction to the choice of ℓ_{max}. Therefore, during the post-ℓ_{max} activities at $S(\ell_{\text{max}})$, one of the following must happen: at least one LSA with a timestamp-source pair greater than $(T(\ell_{\text{max}}), S(\ell_{\text{max}}))$ is received (and accepted), or the TRUE value in the *make_proposal_flag* variable forces ReceiveLSA (or TCTimerHandler if required by a hold-down period) to compute and flood at least one topology with the timestamp value R advanced. Since both cases imply the existence of timestamp-source pairs larger than $(T(\ell_{\text{max}}), S(\ell_{\text{max}}))$, they lead to contradictions to the choice of ℓ_{max}, completing the proof. □

Lemma 3 *The topology $P(\ell_{max})$ is consistent at all network switches.*

Proof: Suppose to the contrary that ℓ_{max} is detected to be inconsistent at some switch y. In response to this situation, switch y sets its *make_proposal_flag* to FALSE (at line 15 of ReceiveLSA). In the meantime, the R variable at y is advanced to $T(\ell_{\text{max}})$ (line 19 of ReceiveLSA), the maximum timestamp value in $\mathcal{L}$, prohibiting subsequent LSAs from being accepted at y (line 11). After y's receipt of ℓ_{max}, there can be no local events at y; otherwise, EventHandler would produce GMC LSAs with timestamps larger than or equal to $T(\ell_{\text{max}}) + 1$, a contradiction to the choice of ℓ_{max}.

The TRUE value of *make_proposal_flag* will cause new topology computations at line 23 of ReceiveLSA or line 4 of TCTimerHandler. The results of these computations, in the absence of further local events, could be dropped in response to incoming LSAs, which spawn additional GMC activities. However, since the number of LSAs in $\mathcal{L}$ is finite, eventually the result of a post-ℓ_{max} topology computation will be flooded with timestamp $R+1 = T(\ell_{\text{max}})+1$, a contradiction to the choice of ℓ_{max}. □

Since the GMC LSA ℓ_{max} has the maximum topology ID and includes a proposal that is consistent at all operational network switches, it shall be accepted by these switches. This observation leads us to the next theorem.

Theorem 1 *Given an MC m and a finite, non-partitioning set of events Π, all operational network switches will reach consensus on MC topology with a finite number of MC LSAs.*

5.4.2 The Handling of Memory Overflows

Next, we consider scenarios where one or more network switches run out of memory space and must purge some entries in their local network images, including MC-related entries. Since a switch can always compute a entirely new MC topology if it has the member list of the MC, the loss of MC topology images (that is, the $P(\mathit{Image})$ data structures) will not cause problems. Hence, we are concerned only with the loss of MC member lists. Further, we assume that switches will not purge data structures other than member lists and MC topology images. The assumption is reasonable because those two are the most space-consuming data structures used by the GMC protocol.

We use an example to illustrate the minor extension to the GMC protocol needed to handle losses of member lists. Consider a scenario where a switch x runs out of storage space and decides to purge the member list of an MC m. The lost member list can be re-constructed when a new topology proposal arrives and is accepted. Should this be the case, the temporary loss of the data structure does no harm to the operation of the GMC protocol. The more interesting case is when switch x must propose a topology after its member list has been purged. One solution is to have x create a member list that incorporates only its own membership status (that is, a member list $\{x\}$ if x is a member, or else, an empty list), and propose an MC topology according to this list. The topology will be found to be inconsistent at all other switches that are members of the MC, triggering topology proposals computed at these switches.

Actually, the GMC protocol with the above revision could survive even more adverse scenarios than isolated, temporary losses of member lists. To investigate tolerance limit of the protocol on this issue, we establish a sufficient condition for the GMC protocol to converge in presence of member list losses.

Lemma 4 *Given a set of events* Π *and an MC* m*, let* M *be the set of members of* m *after* Π*. If there exists at least one switch* $y \in M$ *that does not purge the* Members[m] *data structure indefinitely, then the set* $\mathcal{L}_m$ *is finite.*

Proof: If $\mathcal{L}_m$ is infinite, there must exist switches that indefinitely flood GMC LSAs pertaining to m. Let X be the set of indefinitely flooding switches, and let τ be a moment in time after y has stopped purging *Members*[m] and after all events in Π have been learned by all switches. Define time $\tau' \geq \tau$ to be a moment in time after switch y has constructed the membership status about switches in X (via the infinite number of LSAs from these switches) and after all switches not in X have stopped flooding. If switch y proposes a topology after time τ', then switches in X shall no longer produce triggered LSAs, a contradiction to the selection of X. It can be concluded that y must not be in X, and hence remains silent after time τ'. (The possibility for y to flood an LSA without a topology proposal is excluded, because LSAs without proposals must be event LSAs, which do not exist after time τ.)

If y is to remain silent, all the LSAs produced by some switch in X after time τ' must be consistent at y. To remember the fact that y is a member of m, switches in X cannot purge *Members*[m] after time τ'. Let time $\tau'' \geq \tau'$ be a time after all switches in X have stopped purging *Members*[m], and let us consider any switch $x \in X$. When other switches in X learn the status of x at sometime after τ'' (recall that x floods indefinitely and these switches have plenty of opportunity to learn this information), they shall not subsequently purge it. Subsequent LSAs will be consistent at x, so x will become silent. Therefore, x cannot be in X, a contradiction. We are done. □

The next lemma is a counterpart of Lemmas 2 and 3 combined, in presence of member list losses.

Lemma 5 *Given a set of events* Π *and an MC* m*, let* M *be the set of members of* m *after* Π*. If there exists at least one switch* $y \in M$ *that does not purge the* Members$[m]$ *data structure indefinitely, then the* ℓ_{max} *LSA includes a topology proposal that is consistent at all switches.*

Proof: With a finite set $\mathcal{L}_m$, the maximum LSA, $\ell_{\max}$, in $\mathcal{L}_m$ is well defined. Lemma 2 showed that the $\ell_{\max}$ LSA must contain a topology proposal; otherwise, the switch $S(\ell_{\max})$ would have suggested another LSA with a timestamp larger than $T(\ell_{\max})$. That argument is independent of the issue of member list losses and therefore still holds. However, we need to consider the possibility that the topology $P(\ell_{\max})$ is based on a newly created member list, which could be incomplete when $P(\ell_{\max})$ is computed.

If the member list M' that $S(\ell_{\max})$ uses to compute $P(\ell_{\max})$ is not equal to M, it will trigger LSAs from switches in $M - M'$ and $M' - M$. These switches will be tagged with timestamps larger than $T(\ell_{\max})$, a contradiction to the selection of $\ell_{\max}$. Hence, $\ell_{\max}$ must be based on a complete member list. Lemma 3 guarantees the correctness and network-wide acceptance of its topology proposal, concluding the proof. □

Hence, no LSAs will be able to override the maximum LSA, $\ell_{\max}$, which shall be the consensus on the MC m, even in presence of member list losses. This leads us to the following theorem.

Theorem 2 *Given a set of events* Π *and an MC* m*, let* M *be the set of members* *of m after Π. If there exists at least one switch $y \in M$ that does not purge the* Members$[m]$ *data structure indefinitely, then the GMC protocol will achieve consensus on the topology of the MC m using a finite number of GMC LSAs.*

5.5 Performance Evaluation

A major objective of the GMC protocol is to reduce the redundancy in topology computation incurred by previous LSR-based solutions, while retaining the advantages of LSR (responsiveness, fault tolerance, and so on) and supporting a variety of different MC types. In situations where events are relatively sparse, when a switch detects an event, the GMC protocol suggests a new topology and advertises the topology in an LSA, which will be accepted by all other switches. In this case, there is only one topology computation and one flooding operation per event. This compares very favorably with the MOSPF protocol, which requires a topology computation at every switch involved in the MC. However, it is also important to study the behavior of the GMC protocol when several events occur within a short period of time, during which switches detect inconsistencies in topology proposals and are triggered to prepare and advertise their own proposals. Such situations raise the concern of cascading reactions among switches, which could decrease the advantage of GMC over other approaches. A simulation study was conducted to investigate the behavior of the GMC protocol under such circumstances. The simulator is based on the CSIM simulation package [62].

5.5.1 Simulation methodology

Each simulation session is defined by a set of parameters, including topology computation time, LSA transmission time, event generation distributions, network size, and so forth. In this section, we discuss the selection of parameter values.

We use the symbol T_c to represent the time to compute a topology, and T_f to denote the *flooding diameter* of the network, that is, the time to complete a flooding operation in the worst case. We define the time $T_f + T_c$ to be a *round* which, as mentioned above, is the amount of time needed to handle sparse events in the GMC protocol.

In the GMC protocol, the value of parameter T_c may vary from MC to MC, depending on the choice of the topology computation algorithm for that MC. In this study, we assume the use of Dijkstra's shortest path algorithm, and measured the execution times of the algorithm (using our random graphs as input) on Sun SPARC-20 workstations. The rationale behind the use of Dijkstra's algorithm is its widespread use in computing source-rooted trees [6] and its applicability to several heuristics for computing shared tree topologies (for example, the core-based tree heuristic [8] and the KMB algorithm [31]). Further, this assumption allows us to directly compare the GMC protocol with the MOSPF protocol, which also uses Dijkstra's algorithm.

To determine T_f values, 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 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.

Networks comprising up to 400 switches were simulated. For each network size, 40 graphs were generated randomly, and two simulation sessions were conducted on each graph. Table 5.1 shows the characteristics of the graphs generated. In the table, maximum, minimum, and mean values are averages over the 40 graphs of that size.

The durations of hold-down intervals are uniformly distributed, and are selected randomly each time a hold-down timer is set up by a switch. We investigated the performance of the GMC protocol with no hold-down timers (that is, hold-down intervals are of length zero), a short hold-down interval that is distributed from 2 to 10 rounds, a medium hold-down interval from 20 to 100 rounds, and a long hold-down interval from 200 to 1000 rounds. For example, when a round is 10 milliseconds, the

Network size	degree			diameter			T_f (in ms)	round (in ms)
	min	mean	max	min	mean	max		
10	1.675	3.6	5.675	2	3.25	5	3.555	3.621
20	1.25	3.573	6.725	4	4.75	7	5.123	5.225
40	1.025	3.733	8.025	5	6.175	9	6.705	6.892
60	1.025	3.875	8.65	5	6.8	11	7.5	7.776
80	1	3.914	8.925	6	7.075	9	7.867	8.235
100	1	4.12	9.675	6	7.15	9	8.1	8.558
120	1	4.103	9.725	6	7.525	8	8.423	8.999
140	1	4.201	10.225	7	7.725	9	8.64	9.308
160	1	4.220	10.375	7	7.575	10	8.798	9.576
180	1	4.307	10.5	7	7.75	10	8.9623	9.851
200	1	4.289	10.825	7	7.95	10	9.075	10.078
250	1	4.503	11.275	7	7.95	10	9.338	10.666
300	1	4.704	11.725	7	7.975	10	9.465	11.123
350	1	4.873	12.325	7	7.85	9	9.533	11.422
400	1	5.065	12.65	7	7.85	9	9.623	11.829

Table 5.1: Characteristics of randomly generated graphs.

above interval lengths translate into 0.02 to 0.1 seconds, 0.2 to 1 seconds, and 2 to 10 seconds, respectively.

We are interested in two performance metrics: *topology computations per event* and *flooding operations per event.* The first metric reveals the computational overhead incurred by an MC protocol, and the second measures the communication overhead. In the GMC protocol, the two metrics are not necessarily directly proportional to one another, since computed topologies might not be flooded due to the arrival of new LSAs.

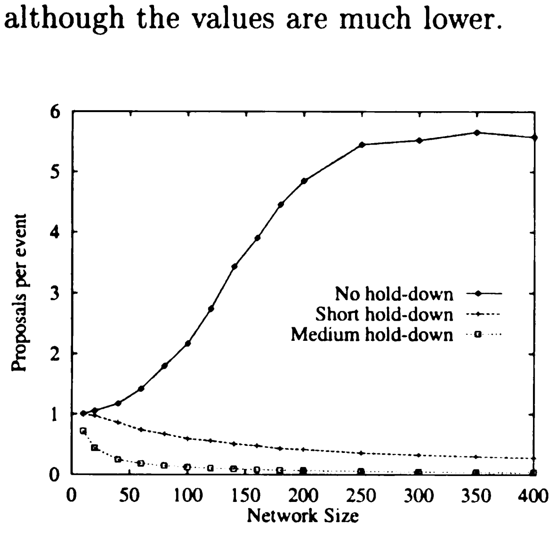
5.5.2 Group Creation Periods

In the first set of experiments, we study the behavior of the GMC protocol during group creation periods. That is, we assume that a group has a predetermined *start time* and that a potentially large number of group members join the group at or about that time. (Such a scenario could occur when, for example, a large number of users join a live broadcast at the beginning of the broadcast.) Specifically, we assume that member arrival times are normally distributed with mean 0, the start time.

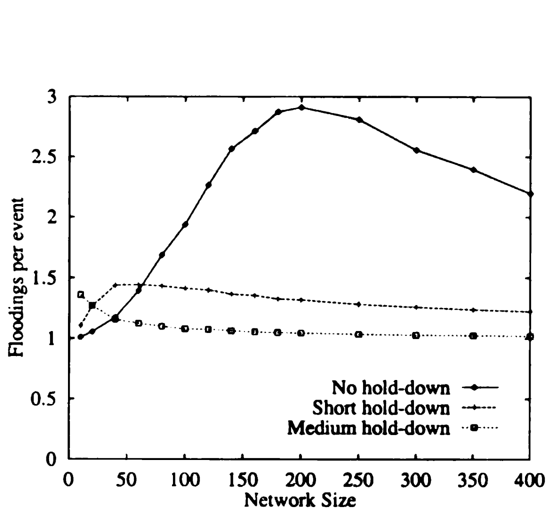
We chose standard deviation values so that 99% of members arrive within a chosen interval length. Specifically, we used standard deviations so that 99% of members arrive within 1 second, 10 seconds, 30 seconds, and 10 minutes. The extremely short creation periods, such as the 1-second and 10-second ones, are designed to stress the GMC protocol during very busy periods. To make the group creation periods as busy as possible, we assume that all switches are group members.

Short arrival intervals. The performance of the GMC protocol with the 1-second member arrival interval is plotted in Figure 5.11. Figure 5.11(a) plots the number of topology computations per event, and Figure 5.11(b) plots the number of floodings per event. When a large number of group members arrive within such a short period of time, cascading interactions among switches could occur if the GMC protocol reacted to events too quickly. This behavior is illustrated by the curves corresponding to no use of hold-down timers. These plots start in the vicinity of one (that is, approximately one computation and flooding per event), because, when the number of group members is small, member join events are still relatively sparse and do not interfere with one another. As the number of switches/members grows and join events collide with each other, these plots reach approximately 5.8 topology computations and 2.9 flooding operations per event, indicating the presence of cascading reactions among switches. However, the curves pertaining to the use of hold-down timers, even a short timer, show that the over-reaction of the GMC protocol can be curbed. With the medium hold-down interval, the number of topology proposals per event approaches zero, and the number of flooding operations per event approaches one. (The latter metric must be greater than or equal to one because the GMC protocol always advertises events immediately, with or without topology proposals.) We note that the number of flooding operations per event when no hold-down is used is not always increasing; see Figure 5.11(b). In general, the number of flooding operations does not necessarily grow with "event density," which is determined by the size of the group when given a fixed arrival interval. This issue will be further addressed later. Results for 10-second and 30-second creation periods are shown in Figures 5.12

and 5.13, respectively. These results are similar to those for the 1-second periods, although the values are much lower.

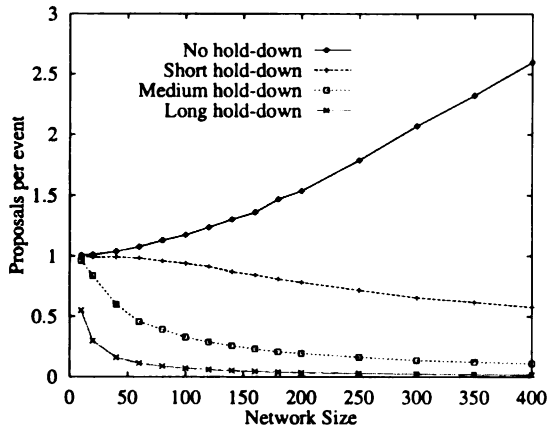


(a) Computations per event.

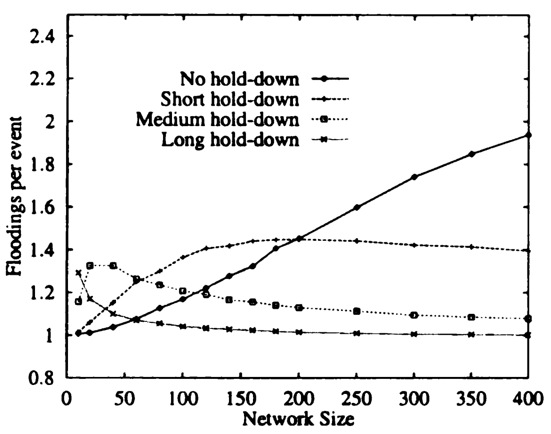


(b) Floodings per event.

Figure 5.11: Performance of the GMC protocol under 1 second arrival interval.



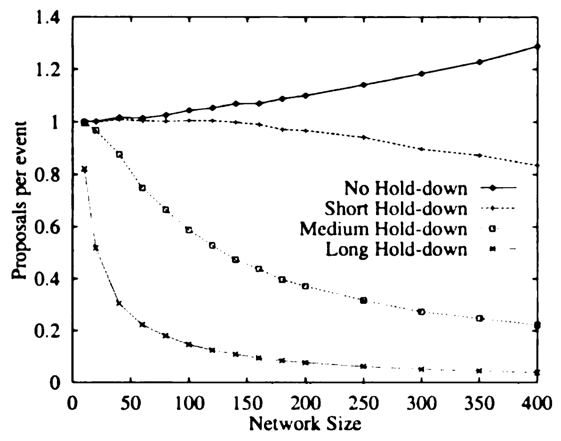
(a) Computations per event.



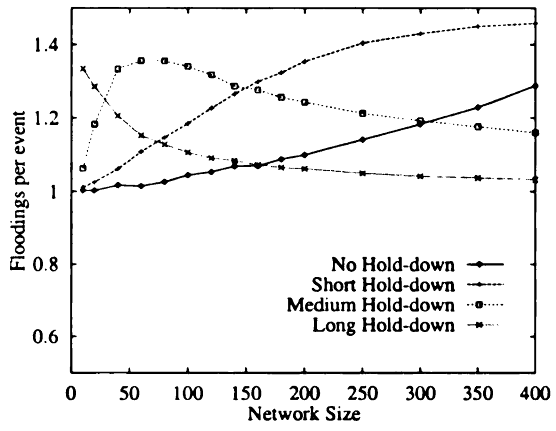
(b) Floodings per event.

Figure 5.12: Performance of the GMC protocol under 10 seconds arrival interval.

10-minute arrival intervals. The performance of the GMC protocol with 10 minutes member arrival interval is plotted in Figure 5.14. With this relatively long arrival interval, the interaction between the event density and the lengths of hold-down intervals becomes clear. When the GMC protocol uses an average hold-down interval



(a) Computations per event.

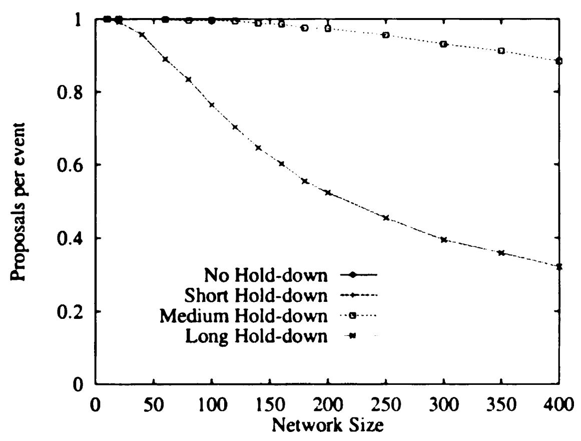


(b) Floodings per event.

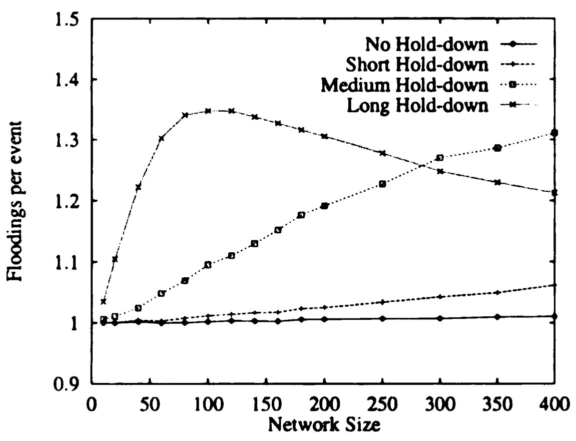
Figure 5.13: Performance of the GMC protocol under 30 seconds arrival intervals.

of Δt seconds, two events must be Δt apart so that their processing does not interfere with each other (otherwise, the second event will be within the hold-down interval created by the first, forcing the GMC protocol to postpone its topology computation). For the short hold-down interval, even the largest networks create "isolated" join events, resulting in the normal operation of the GMC protocol (that is, one topology computation and flooding operation per event), as seen in Figures 5.14 (a) and (b).

With longer hold-down intervals and larger networks, interference among the processing of events can be observed. However, this phenomenon of inter-event interference affects performance metrics differently. Considering the number of topology computations per event, the more inter-event interferences, the more topology computations are suppressed, and hence the fewer topology proposals per event; see the performance results regarding medium and long hold-down intervals in Figure 5.14(a). For flooding operations per event, however, the suppressing of topology computation when the event takes place can introduce later flooding of the delayed topology proposals, resulting in more flooding operations per event. Since the number of topology computations per event decreases as the network size increases, these extra flooding operations (the ones for the delayed topology proposals) become increasingly rare. This behavior is illustrated in Figure 5.14(b) by the curve pertaining to the long



(a) Computations per event.



(b) Floodings per event.

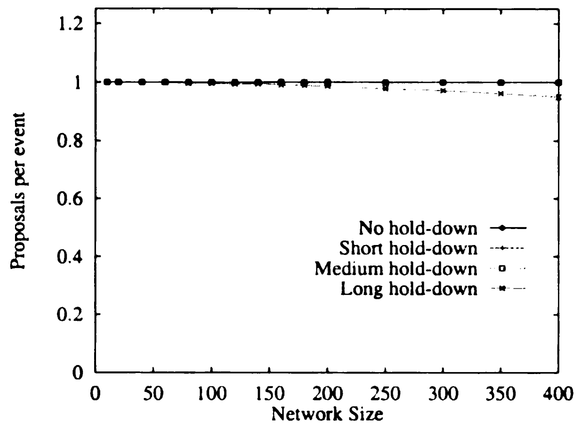
Figure 5.14: Performance of the GMC protocol under the 10 minutes arrival interval.

hold-down interval. A similar phenomenon can be observed in Figure 5.11(b) for the no hold-down case.

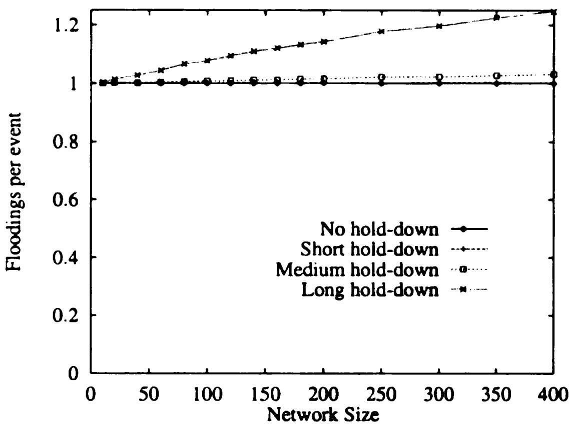
In summary, the high rates of join events during group creation periods can be very demanding of MC protocols. Our simulation results show that, even in extremely busy periods, the GMC protocol is able to avoid cascading protocol activities by using relatively short hold-down intervals (for example, ones less than 0.1 seconds). With longer hold-down intervals, the GMC protocol processes bursty events effectively in batch mode, so that only one topology computation is incurred in response to multiple events. Although this simulation study targets group creation periods, the results apply to any period with high event density.

5.5.3 Normal Operations

During "normal" operation periods of multiparty communication applications, participants join and leave MCs occasionally, and MC protocols may behave differently than during busy periods, such as group creation periods. In this section, we investigate the behavior of GMC under such circumstances. We assume that inter-arrival times of events are exponentially distributed, and we set the event inter-arrival rates in a way such that an N-switch network has approximately $0.3 \times N$ events during a



(a) Computations per event.



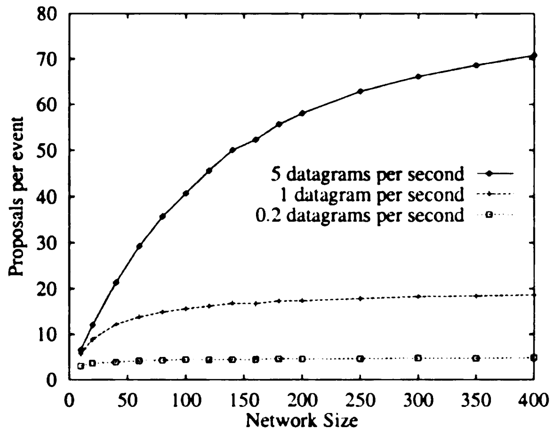
(b) Floodings per event.

Figure 5.15: Performance of the GMC protocol in normal operations.

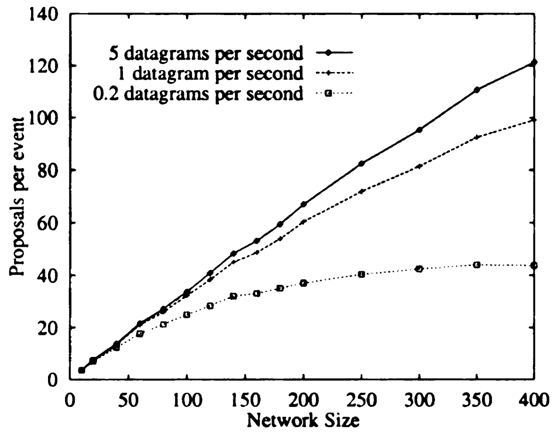
period of 3600 rounds, equivalent to one hour if a round is 10 milliseconds. Under these conditions, it is unlikely (but not impossible) that the processing of one event will interfere with that of the preceding or succeeding events. For such interference to occur, very long hold-down intervals must be used. This behavior is illustrated by the results presented in Figure 5.15, in which only the long hold-down intervals produce slightly less-than-one topology computation per event and more-than-one flooding operation per event.

5.5.4 Comparison with the MOSPF Protocol

The efficiency of the MOSPF protocol depends on three factors: the event rate, the data arrival rate, and the size of the MC. In this protocol, the topology of an MC is cleared whenever an event LSA arrives, and is recomputed when the next datagram for that MC arrives. Once triggered, this computation is performed at all the switches currently involved in the connection. Consider an MC that involves k switches (members and intermediate nodes), and let p be the probability that no datagram arrives between two consecutive events, then the MOSPF protocol would incur $k(1-p)$ topology computations per event. The probability p is determined by the ratio of event arrival rate to datagram arrival rate.



(a) 30-second Creation Interval.



(b) Normal Operation.

Figure 5.16: Topologies computations per event of the MOSPF protocol.

We investigated the performance of the MOSPF protocol during 30-second group creation periods and during normal operation periods, using event distributions as in the GMC simulations and using three datagram arrival rates: 0.2 datagrams per second, 1 datagram per second, and 5 datagrams per second. The results are presented in Figure 5.16. As we can see in Figure 5.16(a), even with a relatively moderate data rate of 5 datagrams per second, the number of topology computations per event can sometimes grow as high as 70 during 30-second creation periods and 120 during normal operation periods. Even with the low data rate of 0.2 datagrams per second, the typical number of topology computations per event is somewhere between 4 and 5 during 30-second creation periods, and can be as high as 40 during normal operation periods. We conclude that the MOSPF protocol incurs significantly higher computational overhead than does the GMC protocol. Regarding the number of flooding operations per event, the MOSPF protocol incurs one flooding operation per event in all circumstances. Although GMC can produce a larger number of floodings, we have seen that a hold-down timer can effectively curb this number.

In summary, the GMC protocol incurs far less computational workload at switches than does the MOSPF protocol, during both group creation periods and normal operation periods. This would allow the GMC protocol to sustain a larger number of simultaneous multicast groups. The use of hold-down timers in GMC produces a tradeoff between responsiveness to events and protocol overhead. GMC's ability to survive memory shortage problems is another significant advantage, especially when the number of groups is large and demands on router memory are heavy. Combining all these factors, we conclude that the GMC protocol is more efficient to supporting individual multicast groups, and scales better in terms of group numbers, than does the MOSPF protocol. Further, we emphasize again that the GMC protocol can accommodate topology algorithms other than the Dijkstra's shortest path algorithm.

5.6 Summary

We have developed an LSR-based generic MC protocol that can be considered as a distributed implementation of any MC topology computation algorithm. Its generality stems from the availability of two pieces of information at every switch, network topology information and MC images. Moreover, this generality enables the use of a single protocol for the construction of MCs of different types and optimized for different performance criteria. The correctness of the GMC protocol is established by formal proofs, and the behavior of the protocol is studied through simulation. The results of these simulations show that the protocol is able to efficiently handle bursts of membership changes in "batch mode," dramatically reducing the protocol overheads during busy periods, while retaining its event-driven nature in normal operation periods. The GMC work shows that LSR provides a solid foundation for supporting one important aspect of group communication, namely the construction of multiparty communication channels. In the next chapter, we develop LSR-based solutions for two other facets of group communication, specifically, membership management and leadership consensus.

Chapter 6

Group Leader Election under Link-State Routing

In this chapter, we investigate an issue involved in both LSR and group communication — the leader election problem. To argue for including leader election as a core network service, we identify applications that can benefit from a network-level leader election protocol, including hierarchical LSR, address mapping, and multicast. A solution to the problem, called the Network Leader Election (NLE) protocol, is proposed for use in LSR-based networks. 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 is proved formally. A simulation study reveals that the NLE protocol incurs low overhead in handling leader failures and in group creation, and compares favorably with a previous LSR-based election protocol, the ATM domain leader election protocol.

6.1 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, for example, in coordinating access to shared resources [63] and in implementing fault-tolerant objects [64]. Generally speaking, solutions to the problem are distributed "host-level" algorithms that make use of various services provided by the network, 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 [65] and the Ring algorithm [52]; more recent developments are described in [66].

In this chapter, 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, 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. 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 [13]; 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 [67] and the mapping between network addresses and link-layer addresses [68], 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 [4] and PIM [2], 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 only single-point failures, but also component failures that may partition the network, the solution to these problems must also survive these adverse scenarios.

Our proposed NLE protocol is based on LSR. Specifically, the NLE protocol extends LSR to include group-leader binding LSAs, which are used by group members to advertise their choice of leader to the rest of the group. 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 consensus on leader bindings, even in presence of adverse conditions. The efficiency of the protocol stems from the use of timestamps to identify obsolete advertisements. We argue that previous solutions to the network-level group leader election problem either do not meet the stringent fault tolerance criteria discussed above, or are more costly (in terms of bandwidth consumption and switch workload) when compared to the NLE protocol. As an extension to LSR, 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 $S_1, S_2, \dots, S_k$, $k \geq 1$, there will be consensus on the leader 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 established 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. Simply put, the NLE protocol can handle leader failures and work properly under catastrophic scenarios such as network partitioning. Results of a simulation study show that these features can be achieved with minimum protocol overhead.

The remainder of this chapter is organized as follows. The design of the NLE protocol is presented in Section 6.2, and the correctness of the protocol, which is modeled as a consensus problem under LSR, is formally proved in Section 6.3. The performance of the NLE protocol and the ATM domain leader election protocol are compared via simulation in Section 6.4. In Section 6.5, 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, a summary of this chapter is given in Section 6.6.

6.2 The NLE Protocol

6.2.1 Overview

Since some decision-making processes of the NLE protocol, such as the leader selection policy, are application dependent, we discuss the protocol operation in the context of the domain leader election problem. As described in Chapter 2, ATM's domain leader election protocol uses a rank-based scheme to select leader (the switch with the highest *leader priority* becomes the leader). Adaptation of the NLE protocol to other problems is discussed in Section 6.5. The operation of the NLE protocol is summarized as follows.

1. For every group g, each switch x in the network maintains a *leader binding*, denoted as $Binding_x(g)$, 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 sends a QUIT-REQUEST message or 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 that has a higher leader priority

than does the current leader.

8. As with other LSR state information, group bindings are subject to aging. To prevent group bindings from aging out, 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 $Leader_x(g) = x$. At a switch $y \neq x$, the binding $Binding_y(g)$ for such a group g will be aged out if this periodic flooding is not received for a predetermined length of time.

6.2.2 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 6.1 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.

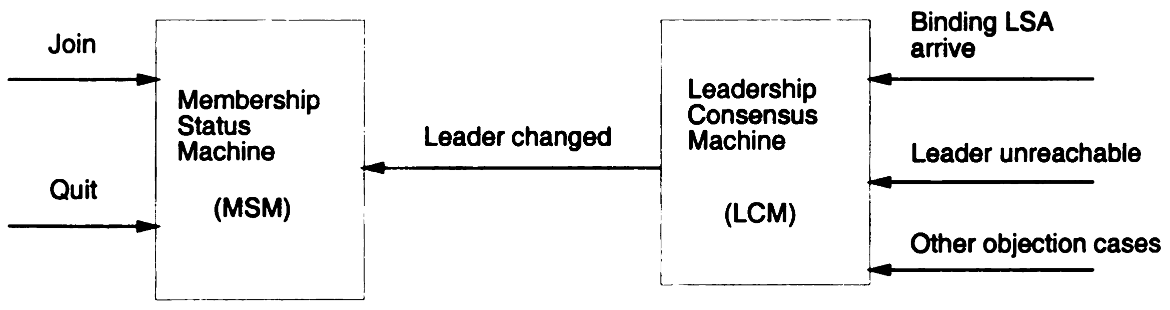


Figure 6.1: The finite state machines in NLE.

6.2.3 The Operation of LCM

The state transition diagram for the LCM is depicted in Figure 6.2. 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.

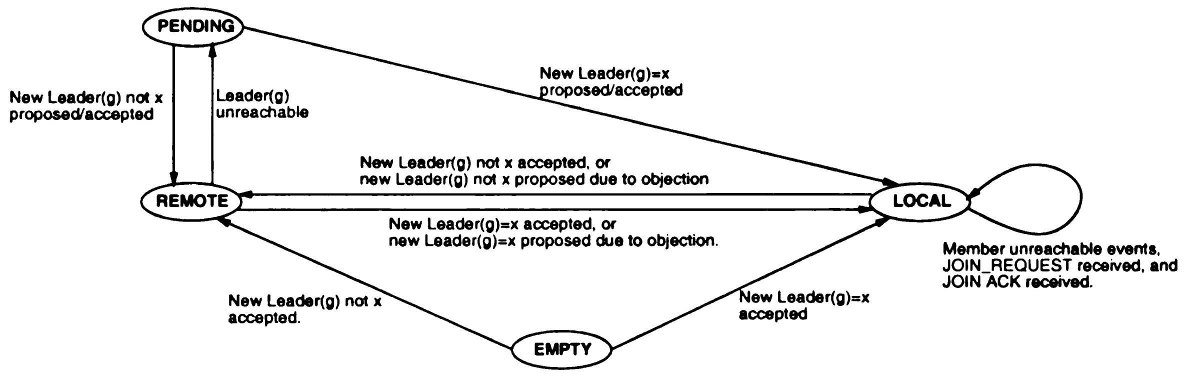

PENDING
New Leader(g) not x proposed/accepted
Leader(g) unreachable
New Leader(g)=x proposed/accepted
New Leader(g) not x accepted, or new Leader(g) not x proposed due to objection
REMOTE
LOCAL
New Leader(g)=x accepted, or new Leader(g)=x proposed due to objection.
Member unreachable events, JOIN_REQUEST received, and JOIN ACK received.
New Leader(g) not x accepted.
New Leader(g)=x accepted
EMPTY

Figure 6.2: 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 6.4.

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.

6.2.4 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.

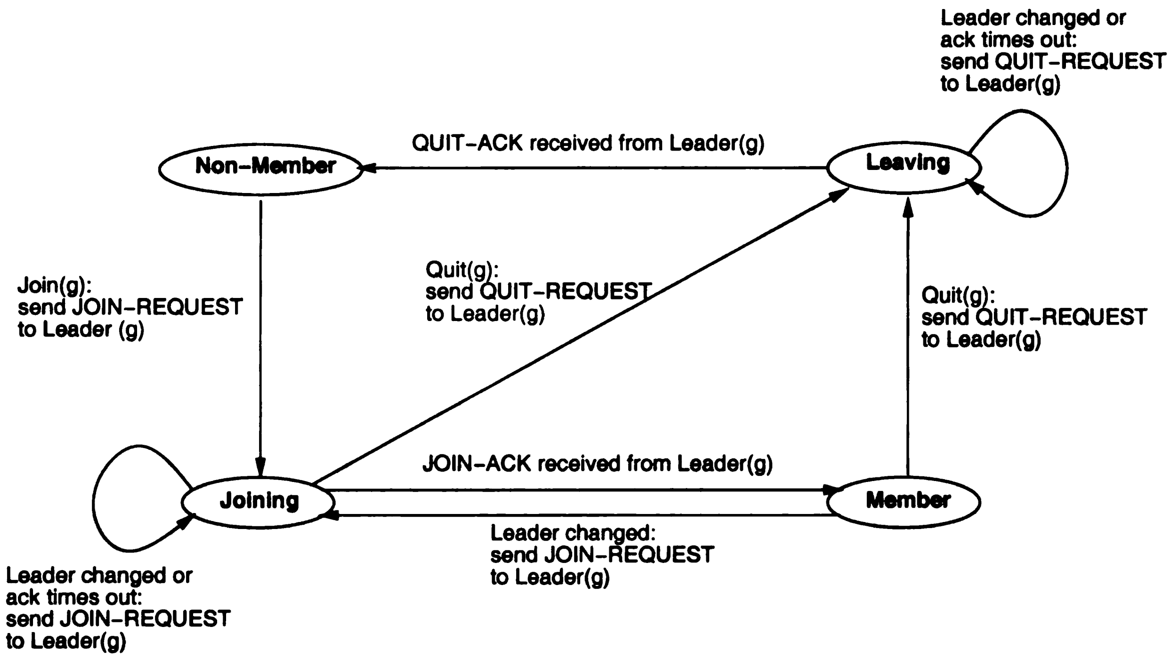


Figure 6.3: The membership status machine at a switch x for a group g (MSM(x, g)).

The MSM does not deal directly with the leader-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).

6.3 Proof of Correctness

We prove in this section that the NLE protocol achieves consensus on group-leader bindings throughout the network. However, we must be careful when defining what can be proved and what cannot be proved. For example, consider a hypothetical scenario where, whenever a switch x is suggested as the leader of a group g, that switch crashes immediately. The other network switches will detect the unreachability to x and (some of them) will propose new leaders. Meanwhile, switch x resumes execution shortly after new leader binding proposals are made. If the scenario repeats indefinitely and every newly suggested leader immediately crashes, then it is impossible for *any* leader-management algorithm to maintain stable and consistent leader bindings for the group.

We conclude that a more reasonable goal is to study the behavior of the NLE protocol in response to a *finite* set of events. (Similar assumptions have been used in the "classic" LSR consensus problem, where the switches must reach consensus on network images [45].) Precisely, let us be given a group g and a finite set of events, E, which may include network status dynamics, group membership changes, unreachability events and other objection events. In addition, E is assumed to contain at least one join event regarding g (otherwise the group is inactive and does not participate in the protocol). We will show that consensus is eventually reached throughout the network on the Binding(g) entries.

We begin with the leadership consensus property. We first prove the property for the case where the network remains connected after the event set E. Following that, we consider the case where the network is partitioned.

Theorem 3 [Weak Leadership Consensus Property] *Let E be a set of network events, including group membership change and network status change events, and let g be a group with at least one join event in E. Assuming that the network is not partitioned after all events in E have taken place, all network switches will eventually agree upon the same leader binding for group g.*

Proof: Considering the set of occurrence times of the events in E, we are interested in the maximum such element, t_{last}, the time of the last event in E. It is to be noted that the assumption of a connected network after time t_{last} does allow for some non-operational switches, as long as the "survivors" remain reachable from one another. Let M_E be the set of switches that are operational and are members of g after E. If M_E is empty, the theorem is true vacuously. Let us consider the more interesting cases where at least one member switch survives E.

Let A be the non-empty set of switches that are operational after t_{last}, and let B be the set of leader binding LSAs for the group g produced in response to the events in E. Let B_A be the subset of B such that $(g, (c, s, t)) \in B_A$ if and only if $c \in A$. In other words, B_A is the set of leader binding LSAs for which the designated leader c is operational after t_{last}.

We claim two properties regarding the set B_A. First, the set B_A cannot be empty, since members of g will select new reachable leaders if there are no valid bindings for g. (A binding is invalid at a switch x if the designated leader is unreachable from x.) Second, the set B_A is finite. In fact, we claim that the set B is finite. This property comes from the fact that the NLE protocol produces a finite number of bindings in response to a single event of any type. The worst case is M binding LSAs per event, where M is the number of members in the group. The worst case happens when the current leader fails and all the M members select/advertise new leaders. Hence, a very loose upper bound for the cardinality of B (and B_A) is $N \times |E|$, where N is the number of switches in the network and an upper bound of M.

Recall that, for two bindings (c_1, s_1, t_1) and (c_2, s_2, t_2), $(c_1, s_1, t_1) > (c_2, s_2, t_2)$ if and only if $(t_1, s_1) > (t_2, s_2)$. Let $b_{max} = (c, s, t)$ be the maximum binding in B_A. This maximum element is well-defined because the set B_A is finite and non-empty. Since the switch c is connected to switches in A after t_{last}, the periodic advertisements of b_{max} from c will eventually be received by all switches in A, which must accept the binding and ignore any others, due to the maximality of b_{max}. The binding b_{max} becomes the final consensus binding among all operational switches, and the theorem is proved. □

The proof of the theorem also suggests that the final leader binding is "correct" in the sense that b_{max} is in B_A (that is, the final winner is an operational switch after E). Somewhat surprisingly, showing consensus when the assumption of eventual network connectivity is removed is not difficult at all, as shown in the following theorem.

Theorem 4 [Leadership Consensus Property] *Let E be a finite event set that partitions the network into k segments, $S_1, S_2, \ldots, S_k$, where $k \geq 1$. Let g be a group with at least one join event in E. The NLE protocol will achieve consensus on leader bindings for g within each segment S_i, for $1 \leq i \leq k$.*

Proof: To see the correctness of the theorem, we apply the argument regarding the set A in the previous proof to each segment S_i. That is, we simply consider switches in S_i to be operational and all other switches to be non-operational. □

Next, we consider the mutual consensus property of the NLE protocol.

Theorem 5 [Mutual Consensus Property] *Given a group g and a set of events E that partitions the network into k segments, $S_1, S_2, \ldots, S_k$, where $k \geq 1$, the consensus leader of g in S_i produces a member list that includes members, and only those members, in S_i.*

Proof: In the following discussion, the consensus leader of g in S_i is denoted as $Leader_i(g)$, and the member list maintained by the leader is denoted as $ML_i(g)$. It is not difficult to see that members not in S_i after E will be removed eventually from

$ML_i(g)$, due to unreachability events about these members. It remains to be shown that all members of g in S_i will be added to the list $ML_i(g)$.

A property of the MSM, shown in Figure 6.3, is that the MSM insists on having the current leader hear about the current membership status. However, some previous membership changes may not be learned by the leader. For example, if a switch x decides to leave a group g while it is in the JOINING state with respect to g, the MSM simply enters the LEAVING state and issues a QUIT-REQUEST; the previous JOIN-REQUEST and JOIN-ACK exchange process is aborted. As a result of this design, given a sequence of interleaved join/quit events, the MSM does not guarantee the success of all respective REQUEST-ACK exchanges, but will enforce the successful exchange with respect to the last event in the sequence.

Let us assume that there is a switch $y \in S_i$ that is a member of g after events in E, but y is not in $ML_i(g)$. By the previous observation, we are concerned only with the REQUEST-ACK exchange process of the last membership change event, which must be a join event. The assumption that y is not in $ML_i(g)$ implies that the leader in S_i does not receive a JOIN-REQUEST message from y, and hence will not return a JOIN-ACK message. Consequently, the switch y remains in the JOINING state, where the JOIN-REQUEST message will be issued repeatedly until corresponding acknowledgment is heard. Since y and $Leader_i(g)$ are connected, this process will eventually complete, putting y on the $ML_i(g)$. This is a contradiction to the assumption about y, concluding the proof. □

6.4 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 [13]. 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 6.1 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. As in the simulations described in previous chapters, we used software overheads of 600 μsec in each LSA forwarding.

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 6.1: 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.

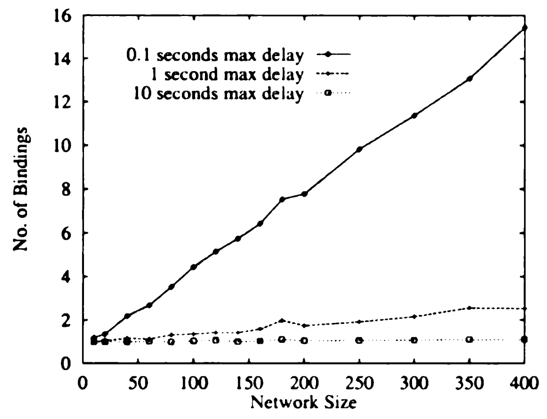
When a leader fails under the NLE protocol, group members select a new leader and send join requests to that switch. Since all members are informed (by corresponding LSAs) almost simultaneously, they all could potentially rush to suggest new leaders, resulting in a large number of conflicting leader binding LSAs. The NLE protocol avoids this problem by deferring member rejoins with a random timer. We assume 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 6.4(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.

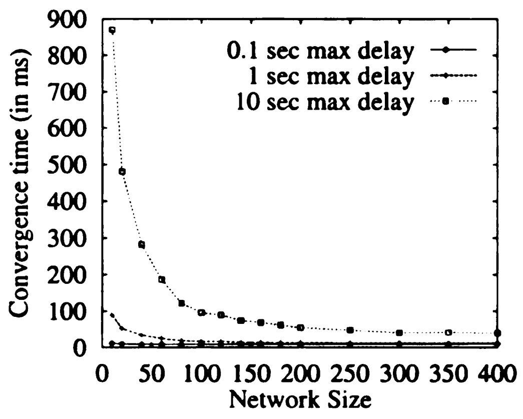
The results for convergence time are plotted in Figure 6.4(b). For this performance metric, the shorter the maximum delay value, the faster the convergence, since a short maximum delay value produces early time-out of the delay timers, and hence switches take less time to flood new leader bindings. Also, 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 6.5(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 6.5(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 6.5(a) conform with those in Figure 6.4(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 6.5(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

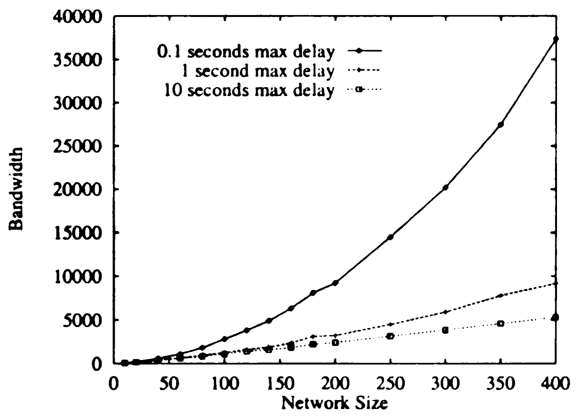


(a) number of bindings.

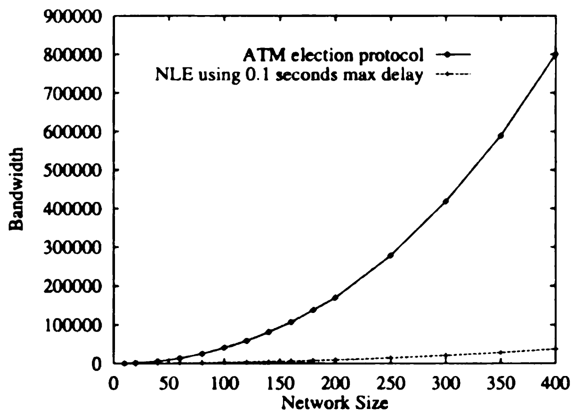


(b) convergence time.

Figure 6.4: Performance of the NLE protocol.



(a) NLE bandwidth.



(b) ATM bandwidth.

Figure 6.5: Bandwidth usage of alternative election protocols.

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.

6.5 Other Potential Uses of The NLE Protocol

We have discussed the use of the NLE protocol for the domain leader election problem. In this section, we briefly discuss the application of the protocol to two other important network services, namely, multicast address resolution and multicast core management. In addition, we evaluate the performance of the NLE protocol in group creation.

6.5.1 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) [67], 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. The approach described in [67] 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 [7]. 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.

6.5.2 Multicast Core Management

As discussed in Chapter 2, some prominent IP multicast protocols, such as CBT [4] and PIM [2], 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.

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 [46]: 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 [46] 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 Chapter 7, 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.

6.5.3 Performance of Multicast Group Creation

When the NLE protocol is used for 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, and group creation time may be important to the performance of those applications.

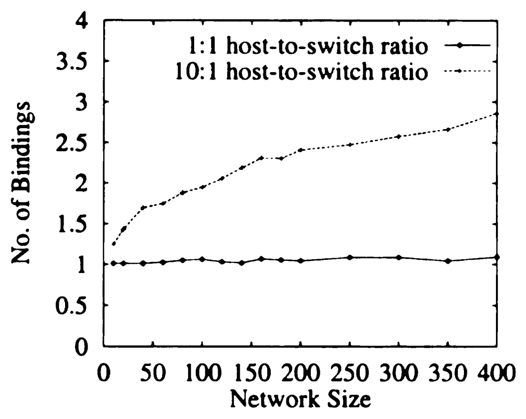
Therefore, we conducted a study to evaluate the performance of the NLE protocol when a multicast group is created. As in the previous performance study, we are interested in two performance metrics: convergence time and the number of binding LSAs. It turns out that the convergence time in this case is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

Theorem 6 *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.*

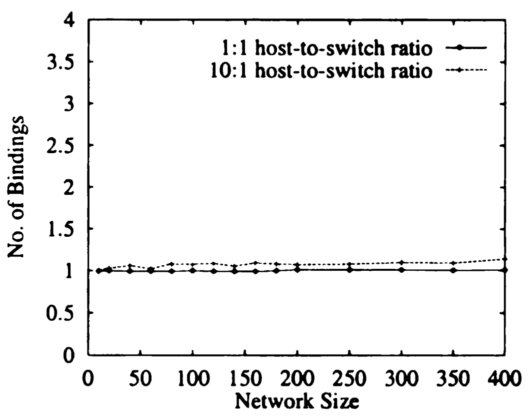
Proof: Assume that the first member joins a group at time 0. This member finds the group unbound and advertises a leader-binding, which will reach all network switches by T_f. Assuming no component failures, any other switch must join the group by time T_f if it is to find the group unbound and propose its own binding. Flooding of any such additional bindings will require another T_f time to finish in the worst case. Therefore, after time $2T_f$, all network switches will have received all the bindings that have been flooded, and will agree upon the one with the largest value. Hence, the worst case convergence time for leader binding is $2T_f$. □

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 6.6 shows the results of this study. Figure 6.6(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 6.6(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 6.6 demonstrate that the NLE protocol is a viable method for handling many real-world situations.



(a) 0.1 seconds arrival intervals.



(b) 1 second arrival intervals.

Figure 6.6: Number of bindings generated for group creation.

6.6 Summary

We have addressed two facets of group communication in LSR-based networks. Specifically, the leader election problem and membership management problem have been studied in a context where participants of the election process are switching elements in LSR-based networks. 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. We have formally proved the correctness of the NLE protocol, including its leadership consensus property and the mutual consensus property, under any combination of group member and network status changes. 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. The efficiency of the NLE protocol enables its use by both the international operations of LSR (such as hierarchical routing and address mapping) as well as multiparty communication applications (for example, those that use core-based multicast). In the next chapter, we propose a second multicast core management method that uses the NLE protocol to select a central server, which manages the core nodes for active multicast groups.

Chapter 7

Multicast Core Management

The problem of multicast core management concerns assigning a network switching element to each multicast group for use as the root of the multicast tree of the group. In the previous chapter, we applied the NLE protocol to this problem in a per-group manner, that is, each group individually holds election to select a respective core node. In this chapter, we pursue an alternative approach to the problem. The proposed method, called the LSR-based Core Management (LCM) protocol, uses the NLE protocol to elect a central server, called the *core binding server* (CBS), to manage core-group bindings for all active groups within the network. The LCM protocol selects core nodes for groups automatically, handles the failures of both core nodes and the CBS itself, supports core migration whereby multicast groups can adapt to membership and network status changes, and survives network partitioning. The LCM protocol is based on LSR: it relies uses the network status information provided by LSR to monitor the operational status of current core nodes and takes advantage of the shortest path trees computed by LSR to support core migration. Our simulation results reveal that the central server can sustain extremely high workloads, and demonstrate the effectiveness of our core selection and core migration methods.

7.1 Introduction

As discussed in Chapter 2, a common technique to support multicast, found in the CBT [4, 3] and PIM [2] protocols, is *core based forwarding* (CBF). A CBF multicast protocol associates a core node with each multicast group; the multicast tree of the group is defined to be the union of core-to-members shortest paths. Messages destined for the group are first sent to the core node, which forwards the message along branches of the tree. An advantage of CBF multicast protocols is that they enable simple methods for nodes to join and leave the group.

We illustrated in Figure 2.3 the member join operation of the CBT protocol. That example assumes that the joining member has learned *a prior* the identity of the core node of the target group. Indeed, many CBF multicast protocols do not concern themselves with core management issues, such as who selects the core node (for example, an administrative authority, users/hosts, or the network), how a core node is selected (that is, which core selection algorithm to use), when a core node is selected (for instance, at the moment a group is created and/or some other time(s) during the life span of the group), how the identity of the core is disseminated to interested parties, and where the identities of the cores of active groups are stored. Before addressing these questions, we identify three basic requirements for core management.

1. **Network-level core selection.** If the task of core selection is performed by hosts, then the multicast interface between hosts and the network depends on the type of the multicast protocol used by the network. (In networks that use a CBF multicast protocol, for example, a join-group request from a host must include the core address of the group, whereas in networks that use other types of multicast protocols such information is not required.) Hence, automatic core selection by the network is preferred over host-level approaches, such as [69].

2. **Core failure handling.** A potential weakness of CBF multicast is the single point of failure at the core. Methods are needed to assign new cores to multicast groups whose current cores have failed.

3. **Core migration.** During the lifetime of a multicast application, the members of a group may change, and the resource availability in the network may fluctuate. The purpose of core migration is to identify a new core node for the group whose corresponding multicast tree, determined by the current set of group members and present network status, will likely result in significantly better multicast performance than the tree based on the current core.

In Chapter 6, we discussed how the NLE protocol can be applied to the core management problem. In that approach, the NLE protocol is applied on a per-group basis to elect core nodes for multicast groups and handle core failures. The practice of storing core-to-group bindings (that is, leader bindings in NLE's terminology) for all the active groups at every router in the network has advantages and disadvantages. On the positive side, the approach adds no additional delays and overheads to group join operations in CBF multicast, because joining members can resolve core-group mappings locally. This merit may be important for multiparty communication applications whose participants join and leave at a high rate. On the negative side, however, the approach raises the concern of scalability when used to support a very large number of simultaneous multicast groups. Alternatively, one could use a bootstrap mechanism, as proposed by the PIM community [33]. In this method, when a multicast group is created or the core node of an existing group has failed, a hash function is used to map the address/ID of the group to a router in the network as its core node. As such, core-bindings need not be stored at all, for all members of a group will map the ID of the group to the same core node. Complexities of the bootstrap mechanism, however, stem from the tasks of discovering and disseminating the identities and operational status of routers in the network. Further, core migration is not supported. To remedy this problem, an independent core migration protocol can be used; at least one such protocol has been proposed by Donahoo et al. [70]. In Donahoo's protocol, the core node of a multicast group periodically sends probing messages to discover a subset of group members and a set of nodes which, if designated as the new core, may improve multicast performance. The core node then sends the list of "representative" members to the selected core candidates, which use sophisticated heuristics to evaluate their performance as the core. Evaluation results are sent back to the current core node, which selects the new core.

In this chapter, we propose a network-level core management method for use in LSR-based networks. The resulting LCM protocol uses the NLE protocol to select a *core-binding server* (CBS), which manages the core-to-group bindings for all active multicast groups within a network. The LCM protocol works closely with LSR, using such information as the identities and operational status of network routers and the topology of the network, to support all three core management issues listed above. A contribution of this work is to demonstrate that a single, and yet relatively simple, core management solution can be developed under LSR.

We emphasize that LSR-based protocols, such as LCM, are not intended for *direct* implementation in very large networks or internets, due to the scalability issues of LSR discussed in Chapter 2. For some CBF multicast protocols, such as the PIM protocol, a core node for a multicast address/group m is assigned within each routing domain that contains at least one member of m. Core management issues under such circumstances are by definition "local;" LCM could be used directly by such protocols. In other CBF multicast protocols, such as the CBT protocol, there is only one core node for a given group m throughout the entire Internet. If the members of m are not restricted to a routing domain, hierarchical core management must be used. In this chapter, we present the "basic" LCM protocol; its extension to hierarchical networks is part of our ongoing research. Hereafter, we use the term "network" to refer to a set of routers that are governed by a single administrative authority and which collectively execute LSR.

The remainder of this chapter is organized as follows. We present the LCM protocol in Section 7.2. Various performance issues, including the workload at the CBS and multicast performance, are investigated through a simulation study, whose results are presented in Section 7.3. These results justify the use of a central server for core management, and show that the performance of multicast can be improved significantly by the simple core migration heuristic supported by LCM. A summary of this work is given in Section 7.4.

7.2 The LCM Protocol

As discussed, the LCM protocol uses a central server, the CBS, to manage core-to-group bindings. Precisely, the CBS of a network maintains a list of core bindings

$$\mathcal{C} = \{\text{Core}(m) \mid m \text{ is an active multicast address}\}.$$

When a host wishes to join to a multicast group m, its local router x sends a CORE-MAPPING(m) message to the CBS. If the binding Core(m) is contained in the list $\mathcal{C}$, then the CBS places this binding in a CORE-ADDRESS message that is returned to x. Otherwise, the CBS selects a core for m according to an *initial core selection heuristic*, and adds this binding to $\mathcal{C}$ before returning the CORE-ADDRESS message. After obtaining the binding Core(m), router x attaches itself to the multicast tree of m using the procedure defined by the underlying CBF multicast protocol. When all attached hosts of router x have departed from group m, router x follows the procedure of the given multicast protocol to exit the multicast tree, without the involvement of the CBS.

Initial core selection. When the first router member of a multicast group m asks the CBS for the core identity of m, group m becomes active and the CBS must select a core node for m. Since at this moment no further membership information regarding m is available, solutions to this initial core selection problem are limited. Previously proposed methods include *random selection* (randomly pick a router in the network) and *random member* (randomly pick a router member) [46]. The LCM protocol adopts a variation of the random member heuristic, called the *first-member* heuristic, which operates as follows: when the CBS receives a CORE-MAPPING(m) request from router x for a group m whose Core(m) does not exist in $\mathcal{C}$, it sets Core(m) to x.

CBS election. The identity of the CBS is not statically configured, but rather is dynamically chosen by the NLE protocol. A specific group identifier (call it CBS-

GID) is reserved for the election of the CBS. Every router x in the network is assumed to be a member of this group, and maintains a leader binding $Leader_x$(CBS-GID), to which CORE-MAPPING messages are sent. The selection and objection policies of the CBS follow a ranking scheme similar to those for domain leader election.

CBS failure handling. To handle CBS failures properly, not only must a new CBS be elected, but also the core binding list $\mathcal{C}$ must be re-collected at the new CBS. For this purpose, each router x maintains a list of core bindings that designate itself as the core, that is, each router x maintains

$$\mathcal{C}_x = \{\text{Core}(m) \mid \text{where } m \text{ is an active multicast address and Core}(m) = x\}.$$

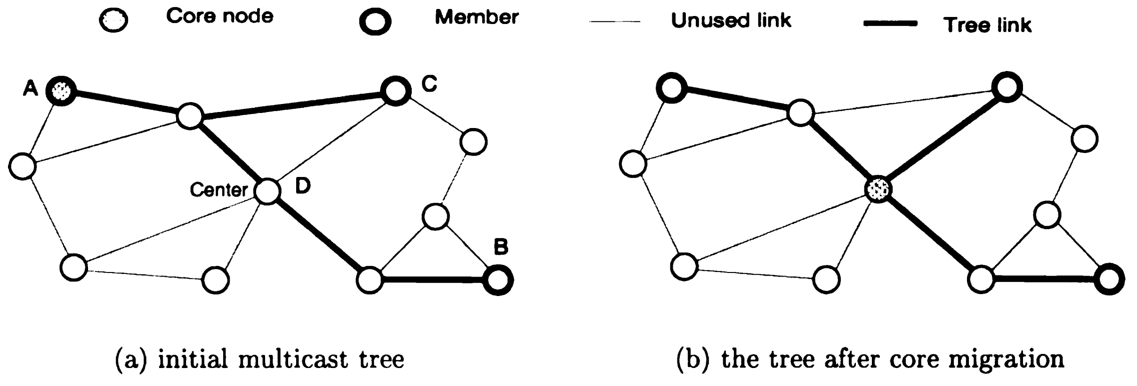
The list $\mathcal{C}_x$ is included in the ballot(s) sent from x during election. Since the CBS will receive ballots from all the routers within the network/segment, it can collect all bindings in $\mathcal{C}$, except those that designated the old CBS as the core of a group, which are discussed below.

Let us consider a network that comprises 4 routers: W (the current CBS), X, Y, and Z. Let $\mathcal{C} = \{(1,W)\ (2,X)\ (3,X)\ (4,X)\ (5,Y)\ (6,Z)\ (7,Z)\}$, where the pair (m,x) denotes Core$(m) = x$. The entire list $\mathcal{C}$ is maintained at W, and the partial binding lists at individual routers are $\mathcal{C}_W = \{(1,W)\}$, $\mathcal{C}_X = \{(2,X)\ (3,X)\ (4,X)\}$, $\mathcal{C}_Y = \{(5,Y)\}$, and $\mathcal{C}_Z = \{(6,Z)\ (7,Z)\}$. Let us assume that router W has failed and that router Y is elected as the new CBS. Since Y will receive partial binding lists, which are contained in respective ballots, from routers X and Z, it reconstructs a new binding list $\mathcal{C} = \{(2,X)\ (3,X)\ (4,X)\ (5,Y)\ (6,Z)\ (7,Z)\}$. However, the core bindings relating to W are missing. To remedy this problem, bindings relating to the old CBS must be treated as a special case. In LCM, any router x that is a member of a group m whose Core(m) = CBS(x) must clear binding Core(m) whenever the value of CBS(x) changes, and must consult the new CBS for a new core binding for m. In the previous example, if group 1 has two members X and Z, then both routers must clear their local Core(1) entries and ask the new CBS Y to provide a new binding for group 1. Of course, such a binding does not exist in the (re-created) binding list $\mathcal{C}$, and consequently the new CBS Y considers group 1 as a newly created group, and uses the initial core selection method to choose a new core node for group 1.

Core failure handing. The CBS uses the network topology information provided by the underlying LSR protocol to monitor all the core nodes listed in $\mathcal{C}$. Specifically, whenever the CBS loses the connectivity to the core node of a group m, it randomly selects a router as the new core of m and advertises this new core binding throughout the network, using the flooding algorithm supported by the underlying LSR protocol. Both the information of router connectivity, required in core failure detection, and identities of routers, required by the random selection heuristic, are made available to the CBS by the underlying LSR protocol. Although our simulation results, presented in the subsequent section, reveal that randomly selecting core node from among all routers typically does not result in good multicast trees, when compared to many other core selection methods, the new core node of the "victim" group can invoke LCM's core migration method, discussed below, to regain (or obtain even better) performance.

Core migration. The core migration method used in LCM assumes that the core node of a group m maintains a (router) member list of m. This list can be compiled and updated if the JOIN-REQUEST and QUIT-REQUEST messages defined by the underlying CBF multicast protocol are delivered to the core node, in addition to the first router on the tree. (The PIM protocol satisfies this requirement. However, minor changes are required for other CBF protocols to meet this requirement.) Periodically, the core node computes a shortest-path tree to reach the members of m, and finds the center of the resulting tree. If the center is not the core itself, the core node voluntarily steps down by sending a CHANGE-CORE message to the CBS, which updates the binding list $\mathcal{C}$ accordingly and floods the new Core(m) value throughout the network. Subsequently, router members of m send JOIN-REQUEST messages to the new core to construct a new multicast tree. We point out that the above shortest-path-tree computation is performed by the underlying LSR routing protocol as part of its normal duties, and that the task of finding the center of a tree can be performed in $O(N)$ complexity, where N is the number of routers on the tree.

As an example, let us consider the network shown in Figure 7.1, where the above core migration method is applied to a group comprising three members A, B, and C. In Figure 7.1(a), we assume that router A is the first node to join the group, and hence is the core of the initial multicast tree of the group. Also shown in the figure is the tree center, router D. In LCM, router A will (eventually) transfer the responsibility of the core node to D, resulting in the multicast tree depicted in Figure 7.1(b). Regarding the performance of the two multicast trees, the tree in Figure 7.1(a) imposes maximum member-to-core distance of 4 and average distance of $(0 + 2 + 4)/3 = 2$, while the tree in Figure 7.1(b) imposes maximum distance of 2 and average distance of $(2 + 2 + 1)/3 = 5/3$.



(a) initial multicast tree

(b) the tree after core migration

Figure 7.1: Core migration in LCM.

Core binding destruction. When the core node of a group m detects an empty group, it sends a DELETE-BINDING message to the CBS, removing the Core(m) entry from the binding list $\mathcal{C}$.

7.3 Performance Evaluation

In this section, we investigate various aspects of the performance of the LCM protocol, including the workload at the CBS, and effectiveness of LCM's core selection/migration policies. Since LCM uses the NLE protocol to select the CBS, simulation results regarding NLE's performance, presented in Chapter 6, apply to CBS election and will be omitted here.

CBS workload. The use of a centralized server for the management of core-group bindings raises the concern of the workload at the server. We investigated this issue via simulation. Our experiments were designed to stress the CBS as much as possible. To this end, we assume that K multicast groups of size S are created simultaneously at time 0. The values of K range from 10 to 200, and those of S range from 20 to 200. Given a multicast group, member arrival times (that is, the times members join the group) are normally distributed with mean 0. We chose the standard deviation value such that 99% of arrival times are within a 1-minute interval centered at time 0 (that is, from -30 seconds to +30 seconds). In the busiest cases, 200 groups of 200 members each are created within 1 minute, producing 40000 CORE-MAPPING requests within that interval. We assumed the service time of such a request to be 700 μsec, which is a typical IP/UDP software overhead observed on many platforms [71]. We used this figure because the look-up of the core binding list can be implemented efficiently, requiring $O(\log S)$ time using a tree-based data structure or $O(1)$ time using a hash function. The overhead of this task should be negligible when compared to the software overhead of receiving and returning messages.

Results of this study are presented in Figure 7.2. As we can see in Figure 7.2(a), the average queue length at the CBS is less than 2, even for the highest event rates. We point out that the queue length is averaged only over the periods where the CBS is busy. Hence, the smallest possible value of the average queue length metric is one. The maximum queue length at the CBS is plotted in Figure 7.2(b). Although the maximum queue length was between 10 and 20 in some experiments, we point out that a queue containing 20 requests can be served within 14 milliseconds. We conclude that the CBS can accommodate even the busiest scenarios in our simulation.

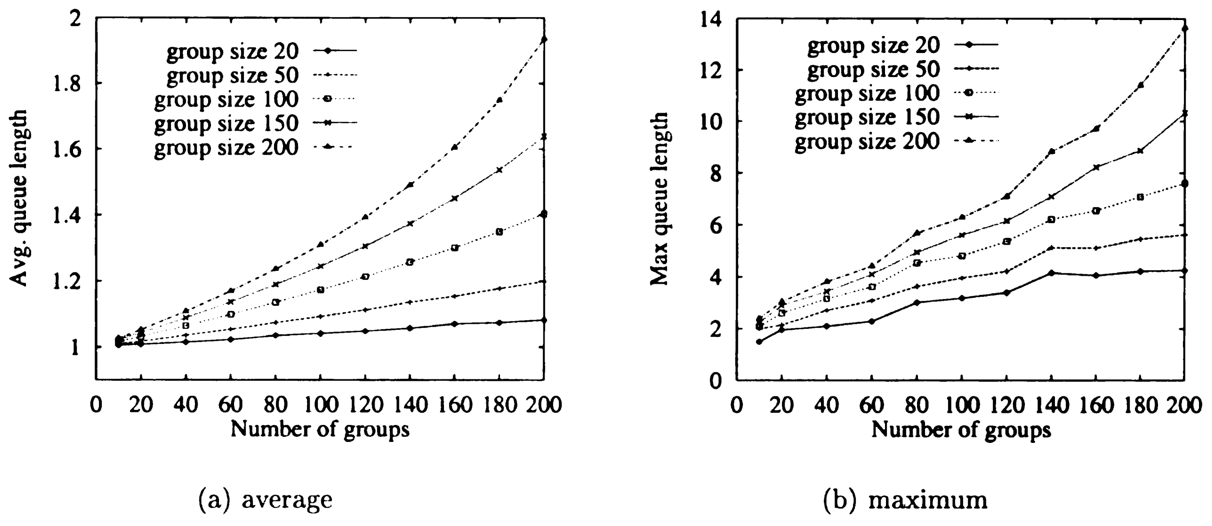


Figure 7.2: Queue length at the CBS.

Core selection/migration. In addition to the operational overhead of the LCM protocol, we also investigated the characteristics of multicast trees that result from the LCM core selection and core migration methods. Specifically, we studied the core-to-member distances of such multicast tress. We randomly generated 100 graphs of 144 nodes (that is, routers) with average node degree 4. The average diameter of these graphs is approximately 10. We randomly generated 1000 groups of size S, where values of S range from 2 to 50. For each group, two multicast trees were generated on each graph. First, a member of the group is randomly selected as the "first member" and is used as the core to construct a multicast tree T. Next, we compute the center of T, which is used as the core to construct a second multicast tree for the group. Furthermore, for each group-graph combination, we tried every the node in the graph as the core node and recorded the average performance of resulting trees, in order to obtain the performance of the random (core) selection heuristic.

The results of average core-to-member distances are plotted in Figure 7.3(a). As we can see, the performance of the first member heuristic is significantly better than that of the random selection method when group size is small, and approaches that of random selection when group size increases. (The flat curve for the random selection method results because the average distance from a group of nodes to a randomly selected node is approximately half the diameter of the network.) Results for the maximum core-to-member distances (that is, the depths of trees) are plotted in Figure 7.3(b). With respect to this metric, the first member and random selection heuristics exhibit approximately identical behaviors. In both Figure 7.3(a) and 7.3(b), the results of the tree-center core selection method clearly demonstrate the benefits of the LCM core migration method, when compared to protocols that do not support migration.

In summary, the results presented here support the core management policies of the LCM protocol, which simply assigns the first member of a multicast group as the initial core node, and changes the core node of the group to the center of the current multicast tree after membership information has been revealed and remained stable for a predetermined length of time.

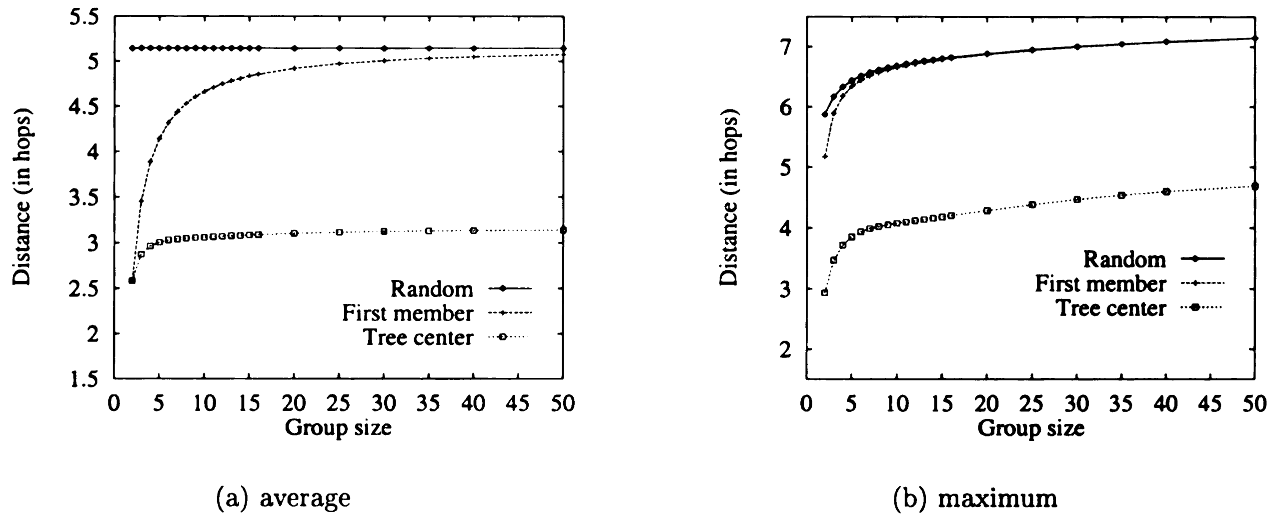


Figure 7.3: Core-to-member distances produced by various core selection methods.

7.4 Summary

We have proposed a central-server based core management protocol, the LCM protocol, for use by CBF multicast protocols under LSR. Based on the information provided by LSR, the protocol addresses three aspects of multicast core management, namely, automatic core selection, core failure handling, and core migration, and can survive any combination of network component failures, including those that partition the network. Our simulation study has shown that the CBS can handle extremely heavy workloads, and has demonstrated the improvements in multicast performance achieved by LCM's core migration method. This work once again illustrates the strength of LSR in supporting group communication.

Chapter 8

Tree-Based Link State Routing

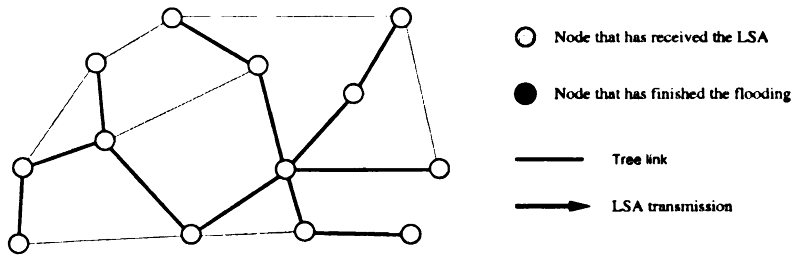
In this chapter, we come full circle, combining group communication techniques discussed earlier to develop a novel link-state routing protocol, called the *Tree-based LSR* (T-LSR) protocol, for use in general-purpose LSR-based networks, such as the Internet. In the T-LSR protocol, a leader router is elected to perform periodic network status broadcast on behalf of all the other routers to reduce the overhead associated with periodic flooding, and a spanning tree is constructed for use by the broadcast of network status updates. We prove the correctness of the T-LSR protocol, that is, its ability to maintain consistent routing information and leader preferences throughout the network under any combination of network component failures, partitioning scenarios, and undetected transmission errors. The results of a simulation study reveal that the T-LSR protocol imposes a small fraction of the overhead of the conventional LSR method during normal operation periods, and incurs moderate overhead during adverse periods when an election is in progress or the spanning tree is under repair/construction.

8.1 Motivation

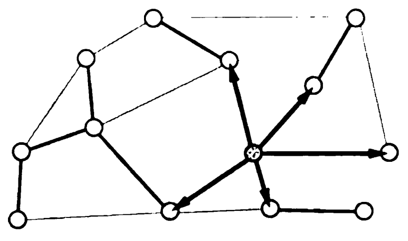
In this chapter, we return to the topic of reducing the operational overhead of LSR. Before presenting our approach, let us take a look again at important performance issues of previous LSR protocols. As discussed earlier, many LSR protocols use the conventional flooding algorithm, which forwards every LSA on every communication link. Thus, each router must process, on average, D copies of a given LSA in a network with average node degree D. Second, all routers are required to flood local status periodically. If the flooding period is T seconds, then each router has to process approximately $(N * D)/T$ LSAs per second in an N-router network. Hereafter, we use the term *conventional LSR*, or *C-LSR* for abbreviation, to refer to any LSR protocol that uses the conventional flooding algorithm and that requires every router to perform periodic flooding. Both the OSPF protocol [11] and the LSR method described in ATM standards [13] fall in this category.

Previous efforts to reduce the overhead of LSR have focused largely on flooding operations. Specifically, Gopal [72] described several hardware implementations of the conventional flooding algorithm. In these implementations, however, a broadcast message still has to traverse all communication links. A software-based, spanning-tree flooding method was discussed in [73]. The main concern of that work was to seamlessly integrate routers that use conventional flooding and with those that use tree-based flooding. It is not clear if that method could survive routing information/transmission corruption problems. Rajagopalan [74] described a flooding method whereby every router builds a source-rooted tree to advertise its local status. By contrast, the T-LSR protocol constructs a single spanning tree shared by every router. Using only one tree reduces the number of protocol states that the underlying flooding algorithm must maintain. Our previous efforts to reduce LSR overhead, namely, the SAF protocols, construct a spanning MC to broadcast LSAs; the idea of hardware-based, spanning-tree broadcast of routing information has also been exploited by other researchers [55, 75]. The T-LSR protocol does not assume any capability in hardware and hence can be applied to a wider range of networking platforms. Furthermore, while the above flooding methods improve the performance of individual flooding operations, none of them are concerned with the bigger picture of the entire flooding cycle.

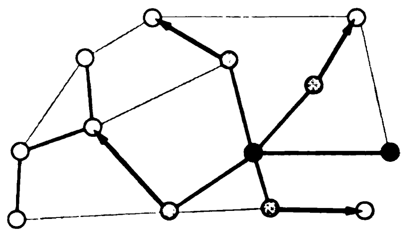
In this paper, we propose a novel LSR protocol, *Tree-based LSR* (T-LSR), which constructs a single spanning tree that is used by all routers for the dissemination of status information. Moreover, the T-LSR protocol elects a leader router to undertake the duty of periodic flooding on behalf of other routers. In Figure 8.1, we give an example to illustrate the concept of tree-based flooding. Using the spanning tree topology shown in Figure 8.1(a), the flooding operation in this example requires four steps. A tree-based flooding operation performs only $O(|V|)$ LSA message forwardings, as opposed to $O(|E|)$ LSA forwardings in the C-LSR protocol. Using the T-LSR protocol, each router in an N-node network that uses T-second flooding cycles processes, on average, only $1/T$ advertisements produced by periodic flooding, and $O(1)$ copies of any LSA.


Node that has received the LSA
Node that has finished the flooding
Tree link
LSA transmission

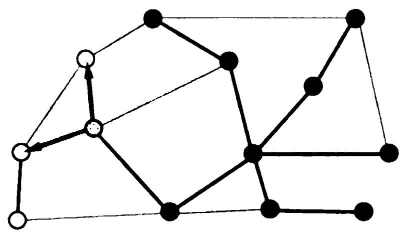
(a) a spanning tree



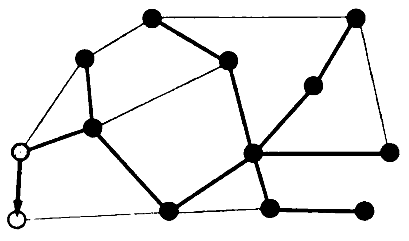
(b) step 1



(c) step 2



(d) step 3



(e) step 4

Figure 8.1: An example of tree-based flooding.

Of course, the major challenge in designing such a "lightweight" LSR protocol is to provide the same level of robustness as the C-LSR protocol. As we discussed in Chapter 2, one of the critical fault-tolerance requirements of an LSR protocol is to survive undetected transmission errors. (The entire ARPANET was brought down by such errors in 1980 [76].) While the problems of leader election and spanning tree construction have been studied extensively [13, 55, 52, 77], previous solutions deal mainly with component failures (such as leader or tree link failures) and partitioning of the network. Solutions to these problems that also survive message corruption events are relatively unexplored. A class of problems, collectively referred to as the *incorrect leadership* problem, arises when corrupted network topology information is used in the computation of the spanning tree topology, or when undetected transmission errors occur during the establishment of leadership and the construction of the spanning tree. We will formally prove the correctness of the T-LSR protocol, that is, its ability to maintain consistent routing information, construct a correct spanning tree, and achieve leadership consensus under any combination of network component failures, partitioning scenarios, and corruption problems.

The remainder of this chapter is organized as follows. An overview of the T-LSR protocol is first given in Section 8.2. Algorithm details of the T-LSR protocol are presented in Section 8.3, followed by the proof of correctness in Section 8.4. The performance of the T-LSR protocol is investigated through simulation. The results of this study, presented in Section 8.5, reveal that the T-LSR protocol imposes a very small fraction of the overhead of the C-LSR protocol during normal operation periods, and incurs only moderate overheads during adverse periods when the spanning tree is under repair/construction and leader election is in progress. Finally, a summarization of this work is given in Section 8.6.

8.2 Overview

In this section, we present the operation of the T-LSR protocol. In the discussion, we assume a connected network $G = (V, E)$, where V is the set of routers and E the set of communication links that connect routers. To generalize our discussion to partitioned networks, we simply consider segments individually. For the purpose of cross reference in the subsequent discussion, important rules/conditions are labeled. (For example, the statement "when a router receives the first copy of a given LSA, the LSA is forwarded along all the links incident to the router except the one on which the LSA arrives" could be labeled as `Forward-LSA-Rule-1`). Before discussion, we give the control messages formats and data structures of the T-LSR protocol in Tables 8.1 and 8.2 respectively.

LSA(x, s, m)	a link-state advertisement with sequence number s and flooding mode m that contains the local status of router x.
CTA$(\alpha, G'_\alpha, T, c)$	a complete-topology advertisement that contains the reachable network image G'_α of the leader router α and a spanning tree topology T with epoch number c.
Ballot(z, α, c)	a ballot message from a child z that specifies α as the leader and is used to establish the spanning tree of epoch number c.
LEA(α, c)	a leadership establishment advertisement that broadcasts the establishment of the leadership of router α and the completion of the construction of the spanning tree with epoch number c suggested by α.

Table 8.1: Control messages in the T-LSR protocol.

Rank(x)	the rank (leader priority) of router x.
Leader(x)	the preferred leader of x.
Mode(x)	the operation mode (either T or G) of x.
Epoch(x)	the epoch number of the current spanning tree.
Flag$_x[z]$	a boolean flag that indicates if x has received the ballot for Leader(x) from its child z in the current spanning tree.

Table 8.2: T-LSR data structures at a router x.

LSA model. We assume that every LSA originated from a router contains complete status of the router. If a router x has five incident links, for example, then every LSA from x contains descriptions of all the five links. When x wishes to advertise the failure of one of its incident links, it floods an LSA that describes the working status of four links and the non-operational status of the fifth. In this way, an LSA can be uniquely identified by its source router ID and a sequence number. This LSA model is similar to that of the OSPF protocol [11]. Other LSR protocols use a more refined model, where each component of the local status of a router (for example, a specific link) is assigned an *LSA ID* [13], and must be identified by a (router ID, LSA ID, sequence number) triple. This allows an LSA to contain a part of the local status of a router and is economical in terms of bandwidth consumption if the router frequently advertises changes in individual state components. The T-LSR protocol could be generalized to handle such LSA models.

Network image. The network image at a router x, denoted as G_x, is defined to be the set of LSAs maintained at x. We note that G_x could include unreachable routers because, for example, when x loses connectivity to another router y, the LSA regarding y will still be maintained by x until it is aged out. We denote by G'_x the set of LSAs maintained at x that are regarding routers reachable from x in the topology defined by G_x. If the network is connected, then $G'_x = G_x$. When the network is partitioned, G'_x is a proper subset of G_x and LSAs in G'_x describe the topology of the network segment in which x resides.

Operation modes. The T-LSR protocol elects a leader router to perform periodic flooding on behalf of all the other routers and uses only tree links in the dissemination of network status updates; details are given later. However, there are periods of time when the election is in progress and/or the spanning tree is under construction. During such adverse periods, the T-LSR protocol reverts to the C-LSR protocol to ensure uninterrupted routing operation. To distinguish adverse periods from normal operation periods, each router operates in one of the following modes: mode T and mode G. We denote by Mode(x) the operation mode at router x.

- During periods when leadership consensus has been achieved and the spanning tree is operational, all the routers in the network operate in mode T. When a router is in mode T, it floods only changes in local status, and uses only spanning tree links in the flooding of LSAs; it does not perform periodic flooding. Every LSA flooded by a T-mode router is tagged with a mode flag of value T; such an LSA is termed *T-mode LSA* and its respective flooding is termed *T-mode* *flooding.*

- When a router is in mode G, it effectively executes the the C-LSR protocol — it performs both periodic and event-driven flooding, which in turn use all communication links. Every LSA flooded by a G-mode router is tagged with a mode flag of value G; such an LSA is termed G-*mode LSA* and its respective flooding is termed G-*mode flooding.* The arrival of a G-mode LSA at a T-mode router forces the router to switch to mode G. The existence of any router in the network that is in mode G indicates a lack of leadership consensus within the network.

Leader election and spanning tree construction. Every router x is configured with a leader priority, denoted by Rank(x), which constitutes a part of the local status of the router, and which therefore is included in LSAs flooded by x. Further, router x searches in $V(G'_x)$, the set of routers known by x to be reachable, for the router with the highest rank, and calls the result of this search its *preferred leader*, denoted as Leader(x). Subsequent actions taken by router x depend on whether or not the value of Leader(x) is x itself.

If Leader(x) is set to x, then router x immediately undertakes the responsibilities of the leader router (although at this point not all routers necessarily agree on its leadership). Leader responsibilities include the computation of a spanning tree topology T and periodic broadcast of *complete topology advertisements* (CTAs). A CTA from x contains all the LSAs in G'_x as well as the spanning tree topology T. To broadcast a CTA, x forwards the CTA along branches of T.

On the other hand, if router x has some other preferred leader, that is, Leader(x) $= \alpha$ and $x \neq \alpha$, then x must await a CTA from α; CTAs from other routers will be silently discarded (`Discard-CTA-Condition-1`). Upon receiving a CTA from its preferred leader, router x processes the LSAs contained in the CTA, extracts the spanning tree T, and forwards the CTA to its children in T. The second task of router x is to receive *ballot* messages for α from all its children. After the completion of this task, x sends its own ballot to its parent y in T. This ballot also serves to establish the x-y tree link. After router α collects all the ballots from its own children, it claims victory by broadcasting a *leadership establishment advertisement* (LEA), again using only T links. Receipt of the LEA changes the operation mode of every router to T, and the network enters the normal operation of the T-LSR protocol.

Re-election. In the T-LSR protocol, leader re-election is triggered by changes in the set of reachable routers. Specifically, when a router x observes a change in the set $V(G'_x)$, it must re-compute its preferred leader (`Compute-Leader-Condition-1`). To enable router ranks to be changed during protocol operation, x also re-computes its leader preference when it detects any change in router ranks (`Compute-Leader-Condition-2`). In either case, router x switches to mode G (`Enter-Mode-G-Condition-1`), and participates in a new election. For illustration, let us consider a network where the administrator has configured a default leader α with rank 3 and a backup leader β with rank 2. All the other routers are configured with rank 1. Consider a scenario where the current leader α has just failed. First, neighboring routers of α notice the failure of links incident to α, and flood LSAs that contain the malfunctioning status of such links. Via these LSAs, every router x detects a change in $V(G'_x)$ (specifically, that α has been removed from $V(G'_x)$), switches to mode G, and sets Leader(x) to the router with the next highest rank, namely β. Router β also discovers that itself is of highest rank, so it broadcasts CTAs and collects ballots to establish its leadership and construct a new spanning tree.

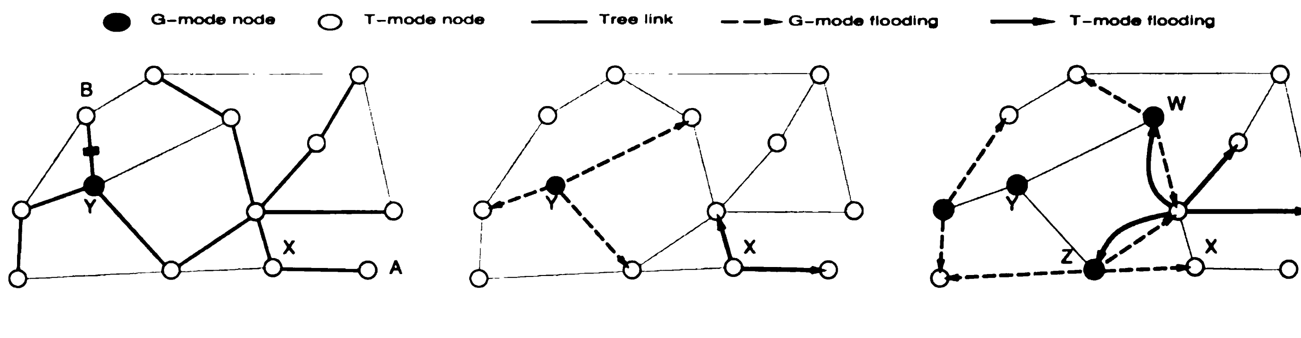
Maintenance of the spanning tree. When a link used by the spanning tree fails, the routers incident to the link switch to mode G (`Enter-Mode-G-Condition-2`) and flood G-mode LSAs that contain the new state of the link. Upon receipt of such an LSA every router in the network switches to G-mode operation. Routers remain in this mode until a new spanning tree (contained in the next CTA from the leader) has been constructed and the leader has broadcast an LEA.

As illustrated above, the spanning tree topologies contained in the periodic broadcast of CTAs from a given leader may change over time in response to network topology changes. The sequence of tree topologies proposed by a leader router is divided into one or more *epochs*. Consecutive, identical tree topologies are tagged with the same epoch number; a change in the tree topology is reflected by an increment in the epoch number. During each epoch, routers remain in mode T. When a change in epoch number is detected, routers switch to mode G (**`Enter-Mode-G-Condition-3`**) until the construction of a new tree is completed. Each router x records the current epoch number in the data structure Epoch(x). Any CTA that contains a spanning tree with a epoch number smaller than Epoch(x) will be discarded by x (**`Discard-CTA-Condition-2`**).

We emphasize that routers must cast ballots in every round of tree topology broadcast (that is, every CTA broadcast), regardless the presence or absence of epoch number changes. Before the broadcast of a CTA, the leader computes a new spanning tree topology, if it is currently in mode G (**`Compute-Tree-Condition-1`**), and increases the epoch number. After receiving all the ballots pertaining to the CTA, if the leader is currently in mode G (**`Issue-LEA-Condition-1`**), it broadcasts an LEA(α, c), where $c =$ Epoch(α). Failing to collect any necessary ballot will switch the leader to mode G (**`Enter-Mode-G-Condition-4`**). Upon receiving the LEA, any router x whose Leader(x) $= \alpha$ and Epoch(x) $= c$ switches to mode T (**`Enter-Mode-T-Condition-1`**) and forwards the LEA to its children in the current spanning tree. Otherwise, the LEA is discarded by x.

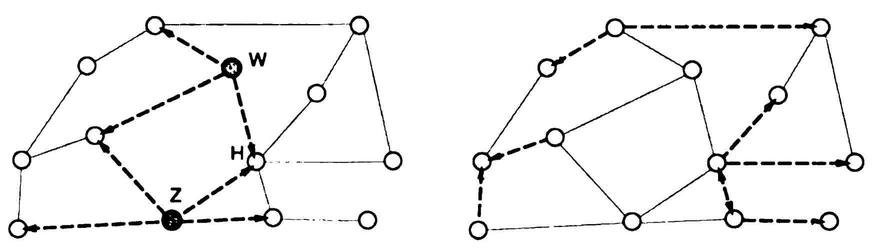
Flooding algorithm. In the T-LSR protocol, a router could operate in mode T or mode G, and an LSA could also be flooded in either one of the two modes. When an LSA arrives at a router, there are four (flooding mode, operation mode) combinations. Before formally presenting the LSA-forwarding rules under these combinations, let us use the example shown in Figure 8.2 to discuss important scenarios. In the example, router X detects a significant change in the queueing delay over the (X, A) link and disseminates this information by flooding an LSA ℓ_X in mode T. Simultaneously, another router Y floods an LSA ℓ_Y (in the G mode) to advertise the failure of the

(Y, B) link, which is used in the spanning tree depicted in Figure 8.2(a). Let us assume that all routers except Y are initially in mode T. Figures 8.2(b) and (c) depict the first and second forwarding steps of the two flooding operations. As shown in Figure 8.2(c), the T-mode LSA ℓ_X encounters two routers, W and Z, whose modes have been changed to G by ℓ_Y. As shown in Figure 8.3(a), when routers W and Z receive ℓ_X, they change the mode of ℓ_X to G and forward ℓ_X along all respective incident links, *including* the ones on which the T-mode ℓ_X arrived, specifically, the (W, H) and (Z, H) links. The G-mode copy of ℓ_X will be considered to be more *recent* than its T-mode counterpart. When arriving at a router that has received the T-mode ℓ_X, the G-mode ℓ_X will be considered being seen for the first time; in Figure 8.3(b), router X forwards the G-mode copy of ℓ_X to its neighbor A, as if it receives ℓ_X for the first time.



(a) initial configuration (b) first forwarding step (c) second forwarding step

Figure 8.2: The flooding of two LSAs in different modes.



(a) first forwarding step (b) second forwarding step

Figure 8.3: The completion of the T-mode flooding in mode G.

However, the above situation where the G-mode copy of ℓ_X returns to X itself raises the concern that ℓ_X may have been corrupted before processed by X the second time. If X blindly accepts the corrupted, G-mode copy of ℓ_X, then router X will have incorrect knowledge about its own status. To cope with problem, when any (G-mode) LSA that contains the local status of a router x arrives at x, router X compares the LSA against its local status and discards the LSA if any inconsistency is detected (`Discard-LSA-Condition-1`).

We now present the flooding rules of the T-LSR protocol. Let LSA $\ell' = \text{LSA}(y, s', m') \in G_x$, where y is the ID of the source router, s is the sequence number, and m' is mode of the LSA, be the LSA regarding y that is maintained at x. When an LSA $\ell = \text{LSA}(y, s, m)$ arrives at x, it is ignored by x if $(s, m) \leq (s', m')$ (`Discard-LSA-Condition-2`), where the comparison is in lexicographic order and mode G is defined to be greater than mode T. (Thus, given two LSAs regarding the same router and with identical sequence numbers, the one in mode G overrides the one in mode T.) If ℓ is not discarded, it substitutes ℓ' in G_x and is forwarded according to the three cases below. In the discussion, we denote by $E(x, \mathsf{T})$ the set of tree links that are incident to x, by $E(x, \mathsf{G})$ the set of incident links of x, and by p the link on which ℓ arrives.

`LSA-Forwarding-Case-1`: $m = \text{Mode}(x)$
    Forward ℓ along links in $E(x, m) - \{p\}$.
`LSA-Forwarding-Case-2`: $m = \mathsf{G}$, and $\text{Mode}(x) = \mathsf{T}$
    Set $\text{Mode}(x)$ to G (`Enter-Mode-G-Condition-5`), and
    forward ℓ along links in $E(x, \mathsf{G}) - \{p\}$.
`LSA-Forwarding-Case-3`: $m = \mathsf{T}$ and $\text{Mode}(x) = \mathsf{G}$
    Forward $\text{LSA}(y, s, \mathsf{G})$ along links in $E(x, \mathsf{G})$.

The first case happens when LSA ℓ and router x are in the same mode. The last two cases take place when the network is in mode transition. In Case 2, the arrival of a G-mode LSA at a T-mode router switches the router to mode G. The LSA itself is forwarded according to the conventional flooding algorithm. In Case 3, when a G-mode router receives a T-mode LSA, that router changes the flooding mode of the LSA to G and forwards the LSA along all its incident links.

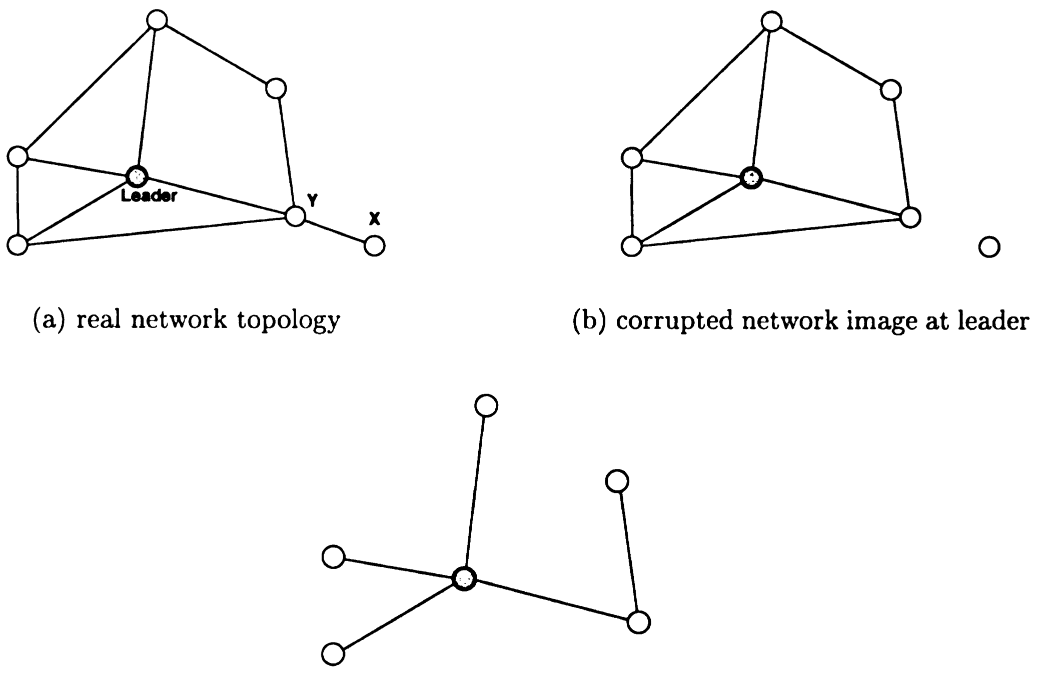
Aging. Like the C-LSR protocol, the T-LSR protocol uses the aging mechanism to curb the lifespan of corrupted LSAs. At a non-leader router x, the LSA in G_x regarding router y will be removed from G_x t_{aging} seconds after its arrival at x. This rule, of course, cannot be applied at the leader router itself, because other routers do not perform periodic flooding and hence the leader may not receive new LSAs from other routers for long periods of time. At the leader router, once leadership has been established, LSAs regarding reachable routers are immune to aging. Specifically, let us consider a given router x and an LSA $\ell \in G_x$ that is regarding another router y. When its associated aging timer fires, ℓ is removed from G_x only if any of the following three conditions are satisfied: Leader(x) $\neq x$ (**`Aging-Condition-1`**), Mode(x) $=$ G (**`Aging-Condition-2`**), and y is unreachable from x in G_x (**`Aging-Condition-3`**). If ℓ is not removed, then a new associated aging timer is created.

However, because a corrupted LSA maintained by the leader regarding a reachable router is not subject to aging, such corruption may exist for prolonged periods of time if not handled properly. Further, the corruption could propagate throughout the network as the leader includes the LSA in CTAs. This problem is detected and corrected as follows. Let ℓ be the LSA in G_α regarding a router x, where α is the leader router. Let ℓ' be the LSA in G_x regarding x itself. When router x receives a CTA from α, which includes ℓ, x checks ℓ against ℓ'. If any inconsistency is found, x switches to mode G (**`Enter-Mode-G-Condition-6`**), discards the CTA (**`Discard-CTA-Condition-3`**), and hence will not vote for α, forcing α also to switch to mode G and thus allowing the corrupted information to be aged out. Meanwhile, router x, now in mode G, floods periodically to provide α with its correct local status information. In order to avoid premature mode switching due to delays in LSAs reaching the leader, the above consistency check is performed only when the CTA is received after $t_{\text{objection\_delay}}$ seconds after the creation of ℓ'.

In the T-LSR protocol, the leadership of the established leader is also subject to aging. Even during periods where there are no network topology changes or when such changes do not affect its leadership, an established leader must periodically flood CTAs that contain the current spanning tree topology and epoch number. If a router does not receive such CTAs for a predetermined length of time, it must revert to mode G operation (`Enter-Mode-G-Condition-7`). Leadership aging addresses the concern where corruption problems in the epoch number of a previous CTA prohibit the acceptance of following CTAs for prolonged periods of time.

The handling of network partitioning. As in the case of handling network component failures, the T-LSR protocol copes with network partitioning scenarios by having every router x monitor the set of routers reachable from x, $V(G'_x)$. Let us consider a scenario where router α is the current leader and a component failure partitions the network into two segments, S_1 and S_2. Let us assume that $\alpha \in S_1$. Routers in S_2 will notice the loss of connectivity to the current leader, switch to mode G (`Enter-Mode-G-Condition-1`), and select a new leader, call it β. Router β will also select itself as the new leader, and, since it is in mode G, will compute and construct a spanning tree within S_2 (`Compute-Tree-Condition-1`). In the meantime, router α will switch to mode G due to changes in the set $V(G'_\alpha)$ and hence will compute a new spanning tree for use in S_1. Should the segments S_1 and S_2 be merged later, routers in S_2, including β, will change their preferred leaders to α. Simultaneously, router α will switch to mode G due to changes in $V(G'_\alpha)$ and hence compute a new spanning tree to cover the entire network.

Handling incorrect leadership problems. As defined earlier, the term incorrect leadership problem refers to any corruption problem involved in leader election and spanning tree construction. Let us consider an example shown in Figure 8.4. In the example, the network image at the leader router α is corrupted in a way that router X, which is reachable from the leader in the real network topology depicted in Figure 8.4(a), is considered unreachable by the leader (see Figure 8.4(b)). Consequently, leader α constructs the incorrect spanning tree T depicted in Figure 8.4(c), which of course does not cover router x. Presuming that every router selects α as the preferred leader, router α will obtain the votes from all the routers covered by T and broadcast an LEA. Subsequently, all routers except X operate in mode T, and any event-driven flooding from a non-X router will use T links and will not reach router X.



(a) real network topology (b) corrupted network image at leader

(c) resultant incorrect spanning tree T.

Figure 8.4: An example of the incorrect leadership problem.

It may be argued that, since X does not receive the above LEA and will remain in mode G, the periodic flooding LSAs from X, which are in mode G, will switch the operation modes of other routers back to G, at least curbing the lifespan of the above situation within a flooding cycle. To see that this mechanism does not necessarily work, let us further assume that $\ell = \text{LSA}(X, s)$ is the LSA in G_X that is regarding X itself and that $\ell' = \text{LSA}(X, s')$ is the corrupted copy of ℓ in G_α. Through the CTA broadcasts from α, LSA ℓ' is propagated throughout the network and incorporated into the network images at all routers but X. If $s' = s + 2^{28}$ and x floods on average every 60 seconds, then it will take X more than 500 years to use sequence numbers larger than s'. Before that, all the LSAs from X are ignored by other routers. The incorrect spanning tree T and the false leadership of router α in Figure 8.4 *can* last for a prolonged period of time.

To cope with this problem, every router, upon receiving a CTA containing a tree topology T, checks whether all its neighboring nodes are present in T. If T fails this test at any router, that router will discard the CTA (`Discard-CTA-Condition-4`) and revert to G-mode operation (`Enter-Mode-G-Condition-8`). In the example of Figure 8.4, router Y shall notice the absence of X from the spanning tree in Figure 8.4(c) and refuse to vote for the leader. This keeps the leader router (and all other routers as well) in mode G, enabling the corrupted information regarding X to be aged out. It is proved in Section 8.4 that, even when the current, incorrect leadership and spanning tree hinder the dissemination of subsequent network status updates, this simple test methodology is sufficient to eventually construct correct leadership and a spanning tree if corruption does not happen to the transmission of T and ensuing ballots indefinitely.

8.3 Algorithms

In this section, we present the algorithms of the T-LSR protocol. In the discussion, for a given router x, we denote by Children(x) the set of children of x in the current spanning tree, relative to Leader(x), and by Parent(x)T the parent of x in the tree.

When a router x needs to flood its local status for the purpose of either periodic flooding or to broadcast changes in its local status, it invokes the FloodLocalStatus routine shown in Figure 8.5. Parameter x in the routine indicates the ID of the caller router. This routine first updates the content of LSA ℓ, the LSA regarding x in its own network image G_x, to reflect the current local status. Next, x switches to mode G and searches for a new preferred leader if there is any change in the set $V(G'_x)$ after the update of ℓ (`Compute-Leader-Condition-1`) or if the rank of x itself has been changed (`Compute-Leader-Condition-2`). Router x must also switch to mode G if any incident tree links are found malfunctioning (`Enter-Mode-G-Condition-2`). Finally, router x increments the sequence number of ℓ and forwards ℓ along the set of links defined by its current operation mode.

Shown in Figure 8.6 is the routine that processes incoming LSAs. In the routine, parameter x indicates the ID of the caller router and ℓ is an incoming LSA

```
Algorithm: FloodLocalStatus.
Input: router ID $x$.

$U = V(G'_x)$.
Let $\ell = \mathrm{LSA}(x, s, m)$ be the LSA regarding router $x$ in $G_x$.
Update the content of $\ell$ (and, hence, $G_x$ and $G'_x$)
        to reflect the current local status of $x$.
IF (Compute-Leader-Condition-1: $U \neq V(G'_x)$) OR
   (Compute-Leader-Condition-2: Rank($x$) has changed) THEN
   Mode($x$)=G. (Enter-Mode-G-Condition-1)
   SetPreferredLeader().
ELSE IF (Enter-Mode-G-Condition-2: $\exists e \in E(x, \mathrm{T})$ that has failed) THEN
   Mode($x$) = G.
   Epoch($x$) = $-1$.
ENDIF
$s = s + 1$.
Forward LSA($x$, $s$, Mode($x$)) along links in $E(x, \mathrm{Mode}(x))$.
```

Figure 8.5: The routine that flood router local status.

that is regarding router y with sequence number s and mode m that arrives on link p. The first task of the routine is to check if x should discard ℓ according to **Discard-LSA-Condition-1** and **Discard-LSA-Condition-2**. Should ℓ pass these tests, it is *accepted* by x and is incorperated into G_x. Subsequently, the ProcessLSA routine checks for changes in the reachable set, $V(G'_x)$, and in the rank of router y. Whenever such a change is detected, router x switches to the G mode and recomputes its preferred leader. Lastly, the routine forwards ℓ according to **Forwarding-Case-1**, **Forwarding-Case-2**, and **Forwarding-Case-3**.

The routine that a router x uses to set its preferred leader, Leader(x), is presented in Figure 8.7. As stated, the preferred leader of x is set to a reachable router w with the highest rank, according to the local network image of x. If x changed its preferred leader, then the current epoch number is set to -1, and, as such, router x can accept a tree topology from the new leader with any epoch number. If the new value of Leader(x) is x itself, the routine invokes the BroadcastCTA routine, discussed next.

When a router x whose Leader(x) $= x$, it periodically invokes the BroadcastCTA routine, shown in Figure 8.8. The routine first checks the ballots corresponding to the previous CTA broadcast and reverts to mode G if there is any ballot missing

```
Algorithm: ProcessLSA.
Input: router ID $x$ and $\ell = \text{LSA}(y, s, m)$ that arrives on link $p$

$U = V(G'_x)$.
IF ($x = y$) THEN /* Check for corruption in LSAs regarding myself */
   Let $\ell' = \text{LSA}(y, s', m')$ be the LSA regarding router $y$ in $G_x$.
   IF /* Discard-LSA-Condition-1 */
      ($s > s'$) OR (($s = s'$) but ($\ell \neq \ell'$)) OR
      (($s < s'$) and ($\ell'$ has existed for more than $t_{\text{objection\_delay}}$ seconds)) THEN
      Mode($x$) = G.
      Exit. /* $\ell$ is discarded */
   ENDIF
ELSE
   IF (Discard-LSA-Condition-2: $(s, m) \leq (s', m')$) THEN Exit.
ENDIF
Replace $\ell'$ with $\ell$ in $G_x$, and set up an aging timer for $\ell$.

/* Changes in leadership rank or the set of reachable routers ? */
IF (Compute-Leader-Condition-1: $U \neq V(G'_x)$) OR
   (Compute-Leader-Condition-2: the ranks of $y$ differ in $\ell$ and $\ell'$) THEN
   Mode($x$)=G.
   SetPreferredLeader().
END

IF (LSA-Forwarding-Case-1: $m$ = Mode($x$)) THEN
   Forward $\ell$ along links $E(x, m) - \{p\}$.
ELSE IF (LSA-Forwarding-Case-2: $m$ = G, and Mode($x$) = T) THEN
   Mode($x$) = G (Enter-Mode-G-Condition-5), and
   forward $\ell$ along links $E(x, \text{G}) - \{p\}$.
ELSE IF (LSA-Forwarding-Case-3: $m$ = T and Mode($x$) = G) THEN
   Forward LSA($x, s$, G) along links $E(x, \text{G})$.
ENDIF
```

Figure 8.6: Processing incoming LSAs.

(`Enter-Mode-G-Condition-4`). (If this is the first CTA broadcast by x, then the check is bound to fail, a result consistent with the fact that x has not established its leadership and must be in mode G.) If x is in mode G, meaning that it is still establishing its leadership, then it must compute a new spanning tree topology T. The routine then broadcasts a CTA that contains the tree topology T and the network image G'_x. Lastly, the routine clears the Flag data structures and router x will await ballots corresponding to this round of CTA broadcast.

When a CTA(α, G'_α, T, c) arrives at a router x via link p, the router invokes the

```
Algorithm: SetPreferredLeader.
Input: router ID $x$.

$old\_leader$ = Leader($x$).
Let $w$ be the router with the highest rank in $G'_x$.
Leader($x$) = $w$.
IF ($old\_leader \neq$ Leader($x$)) THEN
    Epoch($x$) = $-1$.
ENDIF
IF (Leader($x$) = $x$) THEN
    BroadcastCTA().
ENDIF
```

Figure 8.7: Setting preferred leader.

```
Algorithm: BroadcastCTA.
Input: router ID $x$.

/* Checks the ballots of the previous round of votes */
IF (Enter-Mode-G-Condition-4:
    $\exists z \in$ Children($x$) such that Flag$_x[z]$ = FALSE) THEN
    Mode($x$)=G.
ENDIF

IF (Compute-Tree-Condition-1: Mode($x$) = G) THEN
    Compute a tree $T$ that spans $V(G'_x)$.
    Tree($x$) = $T$, and Epoch($x$) = Epoch($x$) + 1.
ENDIF
Forward CTA($x, G'_x, T,$Epoch($x$)) to Children($x$)T

/* To track ballots for this round of vote, */
Flag$_x[z]$ = FALSE, $\forall z \in$ Children($x$)Tree($x$).
```

Figure 8.8: The BroadcastCTA routine.

ProcessCTA routine shown in Figure 8.9. The CTA is discarded, if it is not from the preferred leader of x (`Discard-CTA-Condition-1`), if it contains an obsolete spanning tree topology (`Discard-CTA-Condition-2`), if the LSA regarding x in the CTA is inconsistent with the local status of x (`Discard-CTA-Condition-3`), or if some neighboring routers of x are absent from the spanning tree T contained in the CTA (`Discard-CTA-Condition-4`). If the CTA is accepted by x, the LSAs contained in the CTA are incorporated into the network image of x. Lastly, if the CTA contains a more recent tree topology than the one stored locally, x updates its Epoch(x) data structure accordingly, and switches to mode G to avoid the use of tree-based flooding (`Enter-Mode-G-Condition-3`). Since routers must cast ballots in every round of CTA broadcast, router x, before ending the routine, sets $\text{Flag}_x[z]$ to FALSE for all $z \in \text{Children}(x)$ to await ballots from its chidlren in the tree T.

```
Algorithm: ProcessCTA.
Input: router ID $x$ and an arriving CTA$(\alpha, G'_\alpha, T, c)$.

IF (Discard-CTA-Condition-1: Leader$(x) \neq \alpha$) OR
   (Discard-CTA-Condition-2: $c <$ Epoch$(x)$) THEN Exit.

Let $\ell =$ LSA$(x, s, m)$ be the LSA in the CTA regarding $x$.
Let $\ell' =$ LSA$(x, s', m')$ be the LSA in $G_x$ regarding $x$.
IF $(m \neq m')$ OR $(s > s')$ OR $((s = s')$ but $(\ell \neq \ell'))$ OR
   $((s < s')$ and ($\ell'$ has existed for more than $t_{\text{objection\_delay}}$ seconds)) THEN
   Mode$(x)$ = G. /* (Enter-Mode-G-Condition-6) */
   Exit. /* (Discard-CTA-Condition-3) */
ENDIF

/* Check for corruption in $T$ */
IF (Discard-CTA-Condition-4: $\exists$ a neighbor of $x \notin V(T)$) THEN
   Mode$(x)$ = G.
   Exit.
ENDIF

FOR (each LSA $\ell =$ LSA$(y, s, m)$, $y \neq x$, contained in the CTA) DO
      Let $\ell' =$ LSA$(y, s', m')$ be the LSA regarding router $y$ in $G_x$.
      IF $((s, m) \geq (s', m'))$ THEN
         Replace $\ell'$ with $\ell$ in $G_x$, and reset the aging timer for $\ell$.
      ENDIF
ENDFOR

Forward this CTA to routers in Children$(x)$T.
IF $(c >$ Epoch$(x))$ THEN
   Mode$(x)$ = G. (Enter-Mode-G-Condition-3)
   Tree$(x) = T$, and Epoch$(x) = c$.
ENDIF
Flag$_x[z]$ = FALSE, for each $z \in$ Children$(x)T$.
```

Figure 8.9: The processing of incoming CTAs.

When a Ballot(z, α, c) message arrives at a router x, the router calls the ProcessBallot routine shown in Figure 8.10. A ballot will be processed only if it is for the perferred leader α of x, if it belongs to epoch Epoch(x), and if it comes from a child of x in Tree(x). To process such a ballot, x sets the flag corresponding to the child z and establishes the x-z tree link. When the Flag data structures indicate the receipt of legitimate ballots from all the children of x, remaining actions of the routine depens on if x is the leader. If Leader(x) = x, then router x issues an LEA(α, Epoch(x)) message to broadcast the successful establishment of its leadership. Otherwise router x casts its own ballot by sending a Ballot(x, α, c) message to its parent.

```
Algorithm: ProcessBallot.
Input: router ID $x$ and a message Ballot($z, \alpha, c$).

IF ($\alpha$ = Leader($x$)) AND ($c$ = Epoch($x$)) AND ($z \in$ Children($x$)Tree($x$)) THEN
   Flag$_x$[$z$] = TRUE.
   Establish $x$-$z$ tree link.
   IF ($\forall z' \in$ Children($x$)Tree($x$), Flag$_x$[$z'$] = TRUE) THEN
      IF (Issue-LEA-Condition-1: Leader($x$) = $x$ and Mode($x$) = G) THEN
         Forward an LEA($x, c$) along all incident links of Tree($x$), if Mode($x$) = G.
      ELSE
         Send message Ballot($x, \alpha, c$) to Parent($x$).
      END
   ENDIF
ENDIF
```

Figure 8.10: The processing of ballot messages.

When an LEA(α, c) arrives at a router x, the router invokes the ProcessLEA routine shown in Figure 8.11. The LEA is first checked for the choice of the leader and the current epoch number. If both conditions are satisfied, then router x switches to mode T and forwards the LEA to its children in the current spanning tree. Hereafter, router x enters the normal operation period of the T-LSR protocol — it stops periodic flooding and will use only tree links to advertise local status updates.

```
Algorithm: ProcessLEA.
Input: router ID $x$ and a LEA($\alpha, c$).

IF (Enter-Mode-T-Condition-1: $\alpha$ = Leader($x$) and $c$ = Epoch($x$)) THEN
   Mode($x$)=T.
   Forward the LEA to all $z \in$ Children($x$).
ENDIF
```

Figure 8.11: The processing of LEAs.

8.4 Proof of Correctness

In this section we prove the correctness of the T-LSR protocol. As with any LSR protocol, we must be careful when defining what can be proved and what cannot be proved. For example, consider the problem of establishing leadership consensus in a hypothetical scenario where, whenever a router α is elected as the leader, that router immediately crashes. The other routers will detect the loss of connectivity to α, prompting a new election. Further, let us assume that router α resumes execution shortly after a new leader is elected. If this scenario repeats itself indefinitely, and every newly suggested leader immediately crashes, then it is impossible for *any* leader-management protocol to maintain stable consensus.

We conclude that a more reasonable goal is to study the behavior of the T-LSR protocol in response to a *finite* set of events. This model reflects real-world circumstances where bursts of adverse events are followed by quite periods, which allow an LSR protocol to return to normal operation. We denote by E a finite set of network status change and transmission corruption events, and by t_0 a time after the last event in E. Let G be the network topology after E and α be the router with the highest rank, Rank(α), in G. Let us assume here that the network topology G is connected. To accommodate disconnected networks, one can simply apply the following argument to individual network segments. It is further assumed that events in E leave the T-LSR protocol in a chaotic state. Specifically, we assume the following at time t_0.

- The elements of network images are assumed to be random. Specifically, at any router x, $V(G) - V(G_x)$ may not be empty (that is, some routers may be absent from G_x) and $V(G_x) - V(G)$ may not be empty ("ghost routers" could exist in the network image of x). Further, the content of the LSA regarding router $y \neq x$ in G_x is also assumed to be random. For example, Rank(y) may be corrupted at x, some incident links of y may be missing in G_x (and hence G_x may not be connected), and ghost incident links of y may exist in G_x.

- At any router x, the values of the Mode(x) and Epoch(x) data structures are assumed to be random.

The goal of this section is to show that the T-LSR protocol will establish correct leadership, construct a correct spanning tree, and achieve consistent network images in the presence of such chaotic states. We do assume, however, that every router x possesses correct knowledge about its own status and local surroundings, specifically, that the LSA regarding x itself is not corrupted in G_x.

We emphasize that the chaotic states described above can only be created by corruption problems, a very rare type of events. Network status changes, a type of events that happen much more frequently when compared to corruption events, will always leave the T-LSR protocol in consistent states. The behaviors of the T-LSR protocol in the handing of network status changes are investigated by a simulation study; results of the study are presented in Section 8.5.

First, we deal with a type of corruption problem, involving ranks, that could hinder the establishment of leadership consensus. Let $\Phi_x(t)$ be the set of routers whose (corrupted) ranks in G_x at time t are higher than Rank(α). There are two possible causes for a router y to be in $\Phi_x(t_0)$: the Rank(y) information is corrupted at x, or y itself is a ghost router. The latter case might happen due to the arrival of an LSA(z, s), whose router ID is corrupted and transformed into a non-existent router ID y and whose rank is corrupted and is larger than Rank(α). Of course, a non-empty $\Phi_x(t)$ will prohibit x from selecting α as its preferred leader. In our first lemma, we show that the set Φ_x will become empty t_{aging} seconds after t_0, where t_{aging} is the length of aging timers.

Lemma 6 *At any router x, $\Phi_x(t)$ is empty at any time $t > t_0 + t_{aging}$.*

Proof: Let y be any router in $\Phi_x(t_0)$. Regarding what may happen to y during the $[t_0, t_0 + t_{\text{aging}}]$ period, there are two cases: First, an LSA or CTA originated at router y might arrive and be accepted during the period, fixing the incorrect rank information regarding y at x and consequently removing y from $\Phi_x(t)$. Second, if neither such LSAs nor CTAs from y are accepted by x during the period, then we claim that the aging mechanism of the T-LSR protocol will remove y from $\Phi_x(t)$. Let

ℓ_y be the LSA in G_x that is regarding y, and let t', $t_0 < t' \leq t_{\text{aging}}$, be the time when the aging timer of ℓ_y fires. Depending on the value of Leader(x) at t', we further consider two subcases. In the discussion, we recall that in the T-LSR protocol all the LSAs maintained by a non-leader router, whose Leader(x) $\neq x$, are subject to aging, whereas at an established leader only LSAs regarding unreachable routers are subject to aging.

1. Leader(x) $\in \Phi_x(t')$. In this case, Leader(x) $\neq x$, because due to the assumption that x possesses correct knowledge of its local status, including its rank information, x cannot be in $\Phi_x(t')$. By **Aging-Condition-1**, ℓ_y ages out, and y is removed from Φ_x.

2. Leader(x) $\notin \Phi_x(t')$. In this case, Leader(x) *may* be x. However, routers in $\Phi_x(t')$, including y, must be disconnected from x in G_x at time t'; otherwise Leader(x) would be set to a reachable router in $\Phi_x(t')$. By **Aging-Condition-3**, ℓ_y ages out, and y is removed from Φ_x.

In any of the above cases, every element $y \in \Phi_x(t_0)$ will be removed from the set by the time $t_0 + t_{\text{aging}}$, concluding the proof. □

The next lemma shows how the T-LSR protocol correctly handles incomplete spanning tree topologies.

Lemma 7 *Given a spanning tree topology T that is broadcast by a router x, if T does not include every router in the network, then T cannot win all the votes for x from the routers covered by T.*

Proof: Let $V_T \in V(T) \cap V(G)$ be the set of routers covered by T, and $V_{\overline{T}} = V(G) - V(T)$ be the set of routers not covered by T. Consider any x-to-y path P in the connected, physical topology G. Since $x \in V_T$ and $y \in V_{\overline{T}}$, there must exist two consecutive nodes w and z in P such that $w \in V_T$ and $z \in V_{\overline{T}}$. If T is discarded by any router in $V(T)$ before arriving at w, then of course T cannot win all the votes for x. Otherwise, when T arrives at w, router w will detect the absence of its neighbor z

in T. By `Discard-CTA-Condition-4`, router w would not vote for x. We are done. □

Next, we investigate what would happen to a leader candidate when some other router do not prefer the candidate.

Lemma 8 *Given any two routers x and y such that Leader(x) $= x$ and Leader(y) $\neq x$ at a time $t \geq t_0$, if Leader(x) is not changed after time t and if Leader(y) is never set to x after time t, then there exists a time $t' \geq t$ such that router x will remain permanently in mode* G *after t'.*

Proof: Since Leader(x) is set to x permanently, router x broadcasts an infinite sequence of spanning trees $(T_1, T_2, T_3, \dots)$ after time t. Let us consider the topology T_1. If T_1 covers y, then, of course, router x cannot obtain the vote of router y. If T_1 does not cover y, by Lemma 7, T_1 cannot win the votes from all the routers in $V(T_1)$. In either case, router x must set its mode to G at a time $t' \geq t$. Because the above argument also applies to every subsequent tree topology T_i, $i > 1$, router x will stay in mode G indefinitely after time t'. We are done. □

With the above properties established, we are ready for the first major result. In the proof, we use the expression "by Lemma 8 (w, z, t)" to cite that lemma with router w in the position of x and z in the position of y, using the time reference point t. Recall that α is assumed to be the router in G with the highest rank.

Theorem 7 (Leadership Consensus Property) *There exists a time $t_1 \geq t_0 + t_{aging}$ such that at any time $t \geq t_1$ and $\forall x \in V(G)$, Leader(x) $= \alpha$.*

Proof: Given a router x, we denote by t_x the earliest time when Φ_x is empty. At router α, Leader(α) at time t_α must be α itself. Moreover, if any router x sets Leader(x) to α at time t_x, then Leader(x) will not be changed after t_x, because no (rank) corruption will occur after time t_0. It follows that Leader(α) will not be changed after t_α.

Next, we consider what happens when a router $x \neq \alpha$ whose Leader(x) is never set to α after t_x. Although Lemma 6 assures us that Φ_x will be empty by time $t_0 + t_{\text{aging}}$, Leader(x) is not guaranteed to be set to α; the rank of α in G_x itself, denoted as $\text{Rank}_x(\alpha)$, may be corrupted. A different preferred leader, whose rank is higher than $\text{Rank}_x(\alpha)$, may be selected by x at t_x. Under such circumstances, by Lemma 8 (α, x, t'), where $t' = \max\{t_x, t_\alpha\}$, by our earlier argument that Leader(α) will not change value after time t_α, and by the selection of x, router α will operate in mode G indefinitely after some time $t'' \geq t'$. Next, we must show that the corrupted $\text{Rank}_x(\alpha)$ information at x is subject to aging, allowing the G-mode periodic flooding from α to correct the corruption.

Let ℓ_α be the LSA in G_x that is regarding router α and that contains the corrupted ranking information $\text{Rank}_x(\alpha)$. Let $\Gamma = (t_1, t_2, t_3, \dots)$ be the sequence of times when the aging timer associated with ℓ_α fires. If there exists any $t_i \in \Gamma$ such that any one of the three aging conditions holds at time t_i, then ℓ_α ages out at time t_i (in this case, t_i is the last, largest element in Γ). Assuming otherwise (that is, none of the three aging conditions holds at every time $t_i \in \Gamma$) would result in an infinite Γ. Under such circumstances, let t_a be the smallest element in Γ that is larger than t_α, the time when Leader(α) is permanently set to α. By `Aging-Condition-1` and the selection of elements of Γ, Leader(x) $= x$ at time t_a. It follows that, by Lemma 8 (x, α, t_a), and the fact that Leader(α) will not change value after time t_a, router x must change permanently to mode G at some time $t'_a \geq t_a$. Let t_b be any element in Γ that is larger than t'_a. At time t_b, Mode(x) $=$ G, a contradiction to the assumption that none of the three aging conditions, including the condition Mode(x) $=$ G, holds at time t_b. Hence, ℓ_α will be aged out, enabling router x to accept the G-mode flooding from α and learn the correct rank of α. Consequently, Leader(x) will be set α, a contradiction to our assumption that Leader(x) will never be set to α. We are done. □

Next, we turn our attention to the problem of achieving consistent routing information. Specifically, we show that all routers will possess network images identical to G. The next lemma establishes this property at the leader router α. In the proof, we use the notation $G_x(t)$ to denote the network image of router x at time t.

Lemma 9 *The network image at the leader α will converge to G.*

Proof: We denote by $\Omega(t)$ the set of routers whose respective information is incorrect in G_α at time t. There are three causes for a router x to be included in $\Omega(t_0)$: x is a ghost router, x is a real router that is absent from G_α, or x is a real router that is present in G_α but whose LSA ℓ_x in G_α is corrupted. We note that new elements (that is, routers) cannot be added to the "corruption set" Ω after time t_0 because corruption problems could not happen after that time. If there exists a time $t \geq t_0$ such that $\Omega(t) = \emptyset$, then G_α converges to G at the same time and will remain so thereafter.

Let us assume the opposite, that is, $\Omega(\infty)$ is not empty. As argued earlier, there exists a time t_α when Leader(α) is set to α permanently, and hence, router α will broadcast, regardless the presence or absence of corruption problems in G_α, an infinite sequence of tree topologies after time t_α. Let us denote by t_Ω the time when the "corruption set" Ω has stabilized to $\Omega(\infty)$. Since incorrect parts in G_α stabilize at time t_Ω, network image G_α itself also stabilizes at that time. This stabilized image will be denoted by $G_\alpha(\infty)$. We further denote by $\mathcal{T} = (T_1, T_2, T_3, \ldots)$ the infinite sequence of tree topologies broadcast by α after time $\max\{t_\Omega, t_\alpha\}$.

First, we claim that there are no ghost routers left in $\Omega(\infty)$. To see this property, we assume that x is a ghost router that remains in $V(G_\alpha)$ indefinitely. If $x \notin V(G'_\alpha(\infty))$, then by `Aging-Condition-3` it will be aged out by time $t_\Omega + t_{\text{aging}}$, a contradiction. If x is in $V(G'_\alpha(\infty))$, then we claim that x is covered by every $T \in \mathcal{T}$. Since x cannot vote, router α will have to remain in mode G indefinitely and age out x, a contradiction.

To see the reason why a ghost router $x \in V(G'_\alpha(\infty))$ must be in $V(T)$ for any $T \in \mathcal{T}$, let us first consider the tree T_1 in $\mathcal{T}$. We point out that the fact that T_1 is used after t_Ω does not suggest it is computed after that time. Therefore, one cannot infer the coverage of x by T_1 directly from the presence of x in $G'_\alpha(\infty)$. Let us assume that x is disconnected from α in G_α when T_1 is computed. It follows that from the point of the computation of T_1 to time t_Ω router α must detect at least one change in the set $V(G'_\alpha)$ (specifically, the addition of x) and must compute a spanning tree to cover x, rendering T_1 obsolete by time t_Ω, a contradiction to the definition of T_1. Hence, T_1 must cover x. Moreover, if router α never re-computes the spanning tree after T_1, then it uses T_1 indefinitely after t_Ω (that is, $T_i = T_1$ for any $i > 1$). If router α does perform this re-computation after T_1, then subsequent tree topologies in $\mathcal{T}$ are based on $G_\alpha(\infty)$ and must contain x. In both cases, every tree $T \in \mathcal{T}$ covers router x.

Next, let us deal with any real router x in $\Omega(\infty)$ whose corrupted LSA ℓ_x remains in G_α indefinitely (that is, ℓ_x is never aged out). Let us consider any tree topology $T \in \mathcal{T}$. If T contains x, then router α needs the vote of x. When T arrives at x, router x will detect the corruption in ℓ_x and refuse to vote for α (**Discard-CTA-Condition-3**), forcing α to switch to mode G. If T does not contain x, then, by Lemma 7, T cannot win all the necessary votes for α and router α must also switch to mode G. Since the above argument applies to all the tree topologies in $\mathcal{T}$, router α will remain in mode G indefinitely. It follows that ℓ_x will age out, a contradiction to the selection of x.

As such, there can be only one type of corruption problems for elements in $\Omega(\infty)$: they must be real routers that are absent from G_α after time t_Ω. Let x be any such a router in set $\Omega(\infty)$. By Lemma 7, router α cannot establish its leadership and will be permanently in mode G after some time t'. By Theorem 7, Leader(x) will be permanently set to α at some time t_x, and thus the LSA regarding α at x must be subject to aging after t_x. This ensures that the G-mode flooding from α will be accepted by x, turning the operation mode of x to G. Consequently, x periodically floods its local status in mode G, which is guaranteed to be accepted by a router α that does not have x in its network image at all. However, this contradicts with the assumption that x is absent from G_α indefinitely. We have excluded all the possible causes of a non-empty $\Omega(\infty)$, and hence have shown that G_α will converge to G. □

After corruption problems in G_α have been "cleaned up," the spanning tree topologies computed by α will be correct and accepted by all the routers in the network, as we show below. Recall that t_α denotes the time when Leader(α) is permanently set to α. Further, since we have shown that $\Omega(\infty) = \emptyset$, t_Ω denotes the time when all the corruption problems in the network image of leader α has been removed.

Theorem 8 (Tree-Topology Consensus Property) *There exists a time after which all the routers in the network agree on the same spanning tree topology, which is a correct spanning tree topology proposed by α.*

Proof:

Let us consider the first spanning tree T_1 in $\mathcal{T}$, the sequence of spanning tree topologies broadcast by α after time $\max\{t_\Omega, t_\alpha\}$. Since T_1 may be computed before t_Ω, that is, before all corruption problems in G_α are resolved, it could contain the three types of flaws listed below.

1. Tree T_1 contains a ghost router x. In this case, of course, x will not vote for α, which must in response switch to mode G.

2. Tree T_1 does not cover a (real) router x. (This case happens because, when T_1 is computed, router x is absent from G_α.) By Lemma 7, T_1 cannot win all the votes from routers in $V(T_1)$ and router α must switch to mode G.

3. Tree T_1 uses a non-existent link x-y, where both x and y are real routers. (This case happens when LSAs regarding x and/or y are corrupted in G_α when T_1 is computed.) Without loss of generality, we assume that x be the parent of y in T_1. If this case happens, since x cannot deliver the CTA that contains T_1 to y, then router y will not vote for α, again forcing α to switch to mode G.

If T_1 suffers from any one of the above flaws, then α switches mode G and must compute a new spanning tree in the next CTA broadcast. The new tree, computed after t_Ω, will be a correct spanning tree for G and be included in CTA broadcasts thereafter. If, on the other hand, T_1 is free from the above problems, then T_1 is a correct spanning tree topology (it covers every router, and does not contain non-existent routers and links), and will be used as the tree topology after t_Ω. In the cases where T_1 is the permanent spanning tree topology after t_Ω, it is worth pointing out that, when T_1 is computed, G_α may still contain corruption problems that do not affect the correctness of spanning tree computation, such as disconnected ghost routers. Further, the above argument does not exclude the possibility that T_1 is computed before t_0. As such, the above argument applies to an empty event set E.

Let T be the final, correct spanning tree topology computed by α and c be the epoch number of T. After all the larger-than-c values of the Epoch(x) data structures throughout the network age out and after all the routers select α as preferred leaders, T will be accepted by all the routers in the network, concluding the proof. □

Finally, we are ready to establish the most important property of any LSR protocol — the capability to maintain correct, consistent network images throughout the network.

Theorem 9 (Network Image Consensus Property) *The network image G_x of every router $x \in V(G)$ will converge to G.*

Proof: After a correct spanning tree is constructed, router α will broadcast an LEA, and all routers will enter mode T operation. Consequently, all non-leader routers stop performing periodic flooding. Since there are no events after t_0, non-leader routers will not flood event-driven LSAs either. Let $\ell = \text{LSA}(y, s, m)$ be the LSA regarding y in G_α and let $\ell' = \text{LSA}(y, s', m')$ be the LSA regarding y in G_x, where x is any non-leader router. Upon receiving a CTA from α, which contains ℓ, if $(s', m') < (s, m)$, then router x will accept LSA ℓ, learning the correct status of y. If $(s', m') \geq (s, m)$, then, since x will not receive periodic flooding from y, ℓ' will be aged out in t_{aging} seconds, again allowing ℓ to be accepted by x. We are done. □

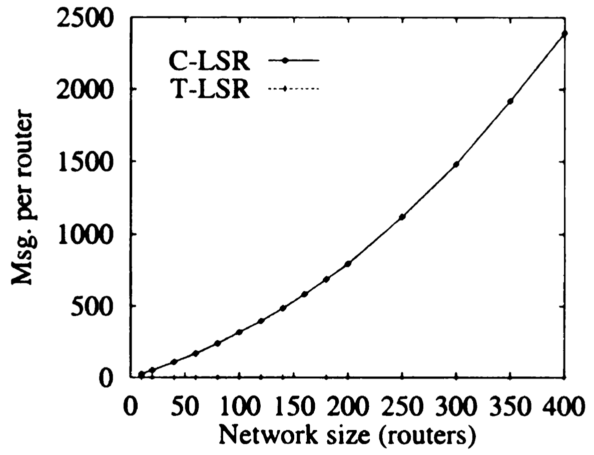
8.5 Performance Evaluation

We studied the performance of the T-LSR protocol through simulation. The simulator is based on the CSIM package [48]. Confidence intervals were computed, but for most cases are very small and, for clarity, are not shown in the plots. Networks comprising up to 400 routers were simulated. Such network sizes conform with those supported by existing LSR standards. (For example, the OSPF protocol supports networks with up to 200 routers.) For each network size, 100 graphs were generated randomly. To conform to network topology characteristics observed in the Internet [57], average node degrees of these graphs are typically small, ranging from 2.25 (for 10-node graphs) to 4 (for 400-node graphs).

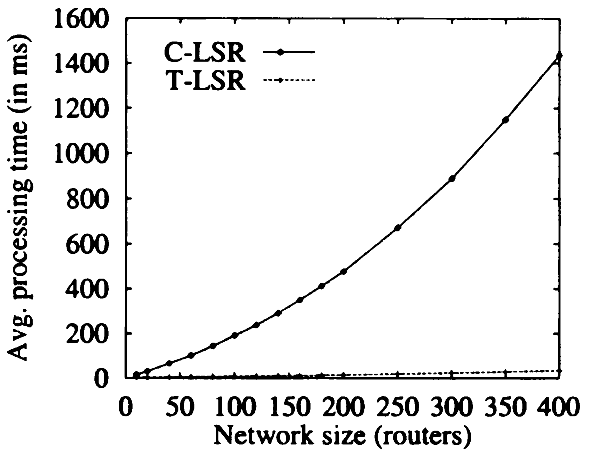
Each message transmission incurs software overheads, including message copying, error checking, processor interruption, and so forth. Of course, such overheads vary from platform to platform. In this study, we measured these overheads on the ATM testbed in our laboratory. The testbed comprises Sun SPARC-10 workstations equipped with Fore SBA-200 adapters and connected with three Fore ASX-200 switches. From these measurements, we obtained the figure 600 μsec, which includes the overhead at both the sending and receiving switches. Since this figure also conforms with typical raw IP overheads that researchers have observed on a variety of workstations [71], the results reported here may be applicable to other LSR-based networking platforms.

Comparison of periodic flooding overhead. First, we compare the T-LSR protocol with the C-LSR protocol when performing periodic flooding. In one cycle of periodic flooding, called a *re-flooding cycle*, each router floods exactly once under the C-LSR protocol. In the case of the T-LSR protocol, the leader router broadcasts a CTA, and other routers cast ballots. For either LSR protocol, we measured the number of messages processed, including acknowledgments, per router per re-flooding cycle. The results of this study, presented in Figure 8.12(a), illustrate a major advantage of the T-LSR protocol, namely, reducing the number of message interrupts at each router.

To account for the differences in message size (the T-LSR protocol uses relatively large messages, namely CTAs, in periodic flooding), we also measured the total message processing time at each router, using the per-byte software overhead 1.09 μs



(a) messages per router



(b) avg. processing time per router

Figure 8.12: Comparison of periodic-flooding overhead.

reported in [71]. The resulting metric can be considered as the average workload at each router. To compute the lengths of CTAs, we used the LSA format of the OSPF protocol, where an LSA of a router with d incident links is $24 + 12 \times d$ bytes long (excluding IP header). Thus, in a network with N routers and with average node degree D, a CTA comprises $N \times (24 + 12 \times D)$ bytes. As we can see in Figure 8.12(b), the T-LSR protocol imposes only a small fraction of the workload of the C-LSR protocol.

To further understand the behavior of the periodic flooding mechanism of the T-LSR protocol, we plot in Figure 8.13 the times used by CTA broadcasts in networks of different sizes. A CTA broadcast begins at the moment when the leader starts sending the corresponding CTA and ends at the moment when the leader receives all necessary ballots. Since, when the leader router is in mode G, CTA broadcast is also used for leader (re)election and spanning tree construction, results in Figure 8.13 can also be interpreted as the leader election and spanning tree construction times of the T-LSR protocol. As we can see in the figure, a CTA broadcast can typically be completed within 350 milliseconds. The overhead of CTA broadcasts primarily stems from the large sizes of CTAs.

Performance of individual flooding operations. In addition to periodic flooding, both LSR protocols use event-driven flooding to disseminate changes in network

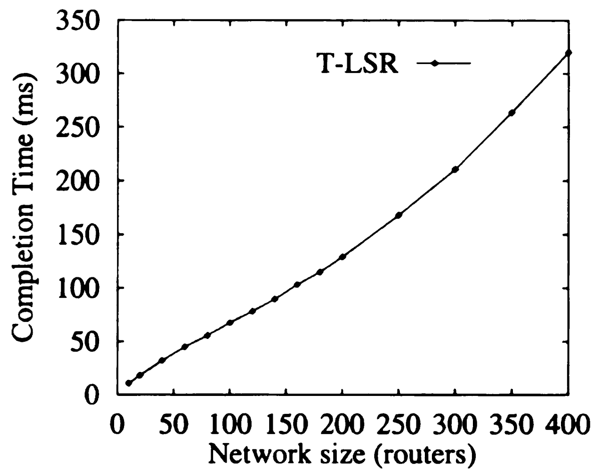


Figure 8.13: Efficiency of CTA broadcast.

status. For such flooding operations, we are interested in three performance metrics: the LSA receipt time, the flooding completion time, and bandwidth consumption. The LSA receipt time of a given router is the time when the first copy of the LSA arrives at the router, whereas the flooding completion time at the router is the time when the router finishes processing the last acknowledgment pertaining to this flooding operation. The bandwidth metric refers to the total number of LSA forwardings incurred by a flooding operation.

The averaged results regarding these metrics are presented in Figure 8.14. As seen, the C-LSR protocol outperforms the T-LSR protocol in both time metrics. This is because, under the conventional flooding algorithm, a router acts aggressively, forwarding an LSA to all its neighboring nodes (rather than only neighbors defined by a spanning tree) and thus causing its neighboring nodes to receive the LSA earlier. However, this aggressiveness also implies that the router has to perform larger numbers of LSA forwarding and process more acknowledgments, as clearly shown in the results regarding the bandwidth metric plotted in Figure 8.14(c). In this metric, the T-LSR protocol enjoys a comfortable lead, of course, because it uses only tree links to forward LSAs.

In summary, during normal operation periods of the T-LSR protocol, a flooding operation is somewhat slower to deliver the respective LSA, but much more economical in terms of operational overhead than its conventional counterpart. Since the

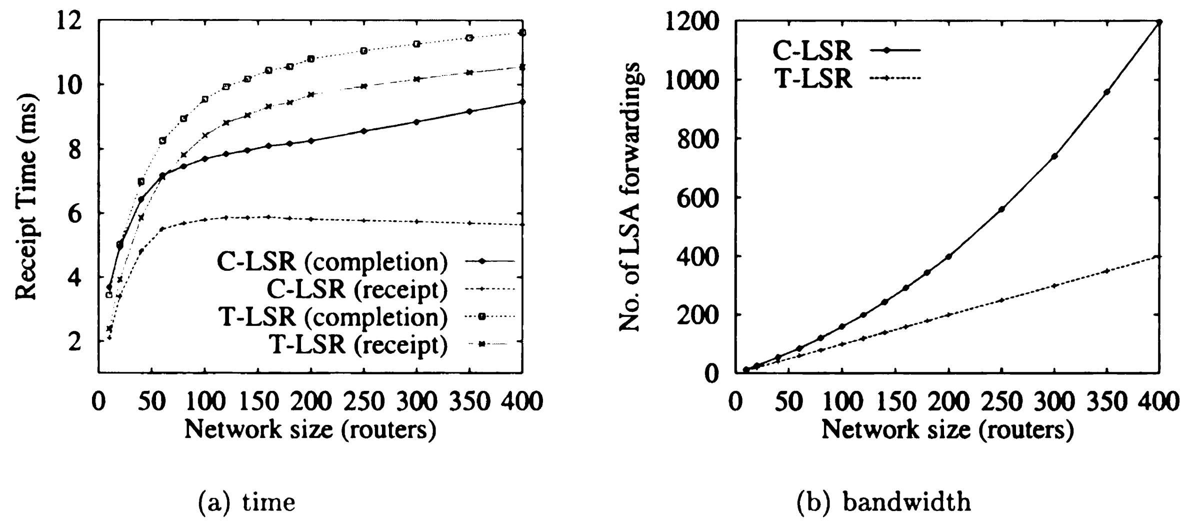
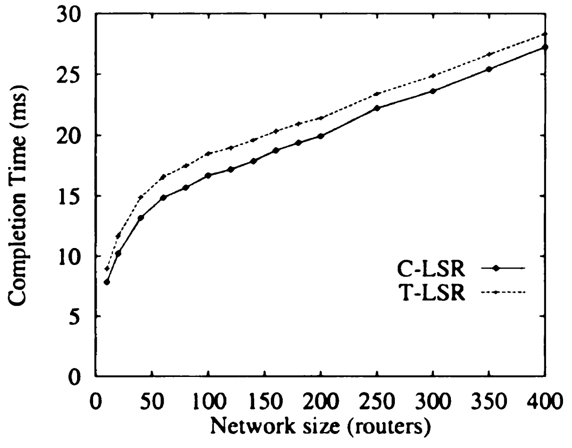


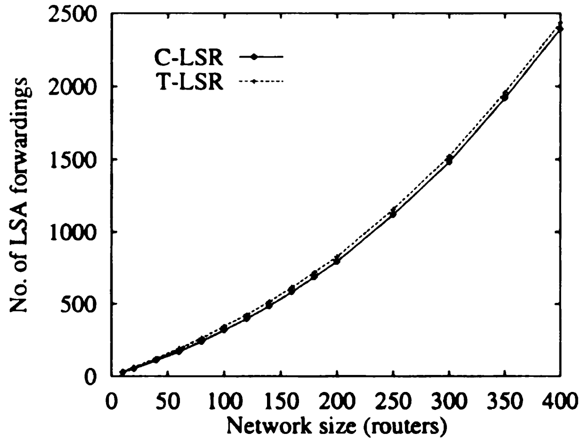
Figure 8.14: Comparison of event-driven flooding performance.

completion times of T-LSR are typically less than 12 milliseconds, the T-LSR protocol still retains the responsiveness of C-LSR. Moreover, in the C-LSR protocol, while an LSA may be received earlier, the ensuing processing of the LSA, such as the updating of routing tables, will be slowed down by the remaining tasks of the flooding operation. It the belief of the authors that T-LSR's large advantage in processing overhead outweighs its slightly slower LSA receipt time.

Evaluation of flooding mode switching. In the T-LSR protocol, a T-mode flooding operation has to switch to mode G if it cannot complete the operation in mode T, for example, due to, the failure of a tree link. In this part of our study, we evaluate the overhead imposed by flooding mode switching. Specifically, we assume that two events e_1, advertised by router x, and e_2, advertised by router y, occur simultaneously, where e_1 affects the spanning tree but e_2 does not. (Prior to the events, all the routers are in mode T.) In the C-LSR protocol, the advertisements of both e_1 and e_2 use the conventional flooding algorithm. In the T-LSR protocol, router x will advertise e_1 in mode G, but router y, without knowing the malfunctioning status of the spanning tree, will initiate the advertisement of e_2 in mode T. The flooding of e_2 will switch to mode G later in order to reach all routers. We simulated these two flooding operations and measured the completion time and bandwidth consumption of the entire "scenario," that is, the flooding of both e_1 and e_2, under different LSR protocols. The respective results are plotted in Figure 8.15. As shown, the T-LSR protocol performs slightly less efficient in both performance metrics. This should not be a surprise, because the advertisement of e_2 in the T-LSR protocol incurs an unfinished T-mode flooding and a complete G-mode flooding.



(a) completion time



(b) no. of LSA forwardings

Figure 8.15: Overhead of flooding mode switching.

We emphasize, however, that flooding mode switching is not expected to occur frequently. For a T-mode flooding to change to mode G, there must be a simultaneous G-mode flooding that has not reached the source of the T-mode flooding. Since the T-LSR protocol advertises in mode G the failure of a network component that damages the spanning tree, according to Figure 6.4(a) such an event can be advertised throughout the network within 10 milliseconds. Thus, only if a T-mode flooding were initiated in the 10 milliseconds window after such a failure would a mode switching occur. As a result, the fraction of T-mode flooding operations that must change mode is likely to be small.

8.6 Summary

We have proposed a novel LSR architecture, called the T-LSR protocol, that elects a leader to perform periodic flooding on behalf of other routers and constructs a spanning tree to reduce of overhead of advertising network status updates. Three consensus properties, namely, the Leader Consensus Property, the Network Image Consensus Property, and Tree-Topology Consensus Property, of the T-LSR protocol under any combination of network component failures, network partitioning, and message corruption events, have been proved formally. Our simulation results show that the T-LSR protocol incurs a small fraction of the overhead of the C-LSR protocol during its normal operation periods, and only moderate overhead in adverse circumstances where the spanning tree is under repair/construction and the leader is being elected. The development of such a lightweight and robust LSR protocol is especially beneficial to communications applications, such as multimedia applications, that demand frequent updates of network status and resource availability information to ensure smooth transit of traffic streams.

Chapter 9

Conclusions And Future Work

If I were to conclude this work in one sentence, I would say that it "re-visits conventional group communication/distributed computing problems under an unusual assumption that complete information about the entire communication network is universally available." Of course, this assumption is not true in general cases; it holds only in a special computing environment where distributed algorithms are executed by routers/switches to implement networking protocols in LSR-based networks. It should not be difficult to see that this computing environment provides powerful facilities that can greatly reduce the complexities of group communication problems. As an example, using the network images maintained by LSR, every participant of an election can learn of the loss of connectivity to the current leader "for free," without using any probing or monitoring mechanism. On the other hand, the low-level nature of this computing environment presents unique challenges. The biggest challenge is that, since networks are expected to continue providing communication services even in the presence of exceedingly rare but catastrophic adverse events, such as network partitioning and undetected transmission errors, the algorithms executed within the network so as to implement these services must also survive such events. A fundamental contribution of this dissertation is to show that developing distributed algorithms specifically for this computing environment results in better group communication solutions and, moreover, improvements to LSR itself.

Using the network images maintained by LSR, we have developed the GMC protocol, which can be considered as a generic distributed implementation of MC/multicast routing algorithms. The ability to support different MC topology types and computation algorithms is important when a wide spectrum of multiparty communication applications, each with unique characteristics and expectations of the network, are deployed.

Using router/switch connectivity information provided by LSR, we have developed a network-level leader election protocol, the NLE protocol. We have discussed important network services that could benefit from the NLE protocol, including hierarchical routing, address mapping services, and multicast. Based on the NLE protocol, we have designed a centralized solution to the problem of multicast core management, namely, the LCM protocol. In addition to using network images, the LCM protocol further makes use of the shortest path routing trees computed by LSR to support certain tasks of multicast core management, such as, core migration.

Finally, one of the most important group communication problems is network routing itself. Since every network switching element can observe only its local surrounding, the task of finding paths to relay communication traffic across the network must be performed by all routers/switches collectively. In this dissertation, we have advocated the use of group communication techniques to improve the performance of LSR. For ATM networks, we have developed a family of efficient flooding algorithms, the SAF protocols, that take advantage of the hardware switching capabilities of such networks. These protocols construct a spanning tree and a ring in a given ATM network to improve the performance of flooding operations in the network. For other LSR-based networks, such as many autonomous systems in the Internet, we have developed the T-LSR protocol to reduce the overhead associated with both periodic and event-driven flooding, using two group communication based techniques, spanning tree construction and leader election. Considering all these results, we have clearly demonstrated the mutually beneficial relationship between group communication and LSR.

The research of this dissertation can be extended in several directions, described as follows.

As pointed out earlier, LSR is not intended for direct implementation in large networks. This restriction inevitably raises the question that how our group communication protocols, which are all LSR-based, can be applied in such networks. In the case of ATM networks, the entire network is recursively divided into smaller routing domains, and the same routing method, namely LSR, is applied at all routing levels. In such circumstances, our group communication solutions can be executed recursively in the routing hierarchy. For example, to construct a receiver-only MC in a large ATM network, a top-level MC can be constructed at the top routing level; members of the MC are representative group members elected in the second highest routing domains that have at least one member of the MC. Subsequently, each such domain constructs a second-level MC within that domain. The low-level MCs and the top-level MC are connected together using the representative group members in low-level domains as contact points. This process is repeated until the lowest routing level is reached. In fact, a well-defined routing hierarchy may enable the use of different MC topology types and computation algorithms at different levels for the same network group. We point out that the PIM protocol already supports such "hybrid MCs," to a limited extent, by constructing source rooted trees at the inter-AS level and shared trees within ASs. The generalization of the GMC protocol to allow any MC type at any routing level and the potential applications of hybrid MCs constitute an interesting area of future research.

However, in other networks, most prominently the Internet, LSR is restricted to individual routing "islands," (that is, routing domains or autonomous systems) and another routing method is used to perform routing among these islands. In such cases, an LSR-based group communication protocol must cooperate seamlessly with a high-level protocol, which may not be LSR-based. Such integration issues require further investigation. For this integration problem, the technique that uses a leader election to reduce an LSR-based domain to a single node could play an important role. Considering again the example of the construction of a network-wide MC, an inter-AS MC protocol can consider an LSR-based AS as a single node by electing a representative member in that domain.

Another important area of future research is the support of QoS routing, which finds paths to carry resource-demanding, multimedia traffic. Many methods developed in this dissertation address the operational aspects of network routing and could be used in the design of mechanisms that timely disseminate the information required by QoS routing. For example, our methods could be used to elect routing server/center, and/or reduce the workload of individual routers/switches. One promising possibility is to elect a leader router to periodically collect and broadcast the resource utilization status of the entire network. This collect-and-broadcast process could be a variation of the CTA broadcast and ballot collection process used in the T-LSR protocol (specifically, each router includes its up-to-date local status in its ballots). When the resource utilization status of the network fluctuates at a high rate, for example, due to a long burst of VC establishment and destruction requests, using an orderly process of information collection and dissemination might produce much more efficient routing operations, when compared to having all routers/switches flood their status changes individually. Furthermore, an adaptive LSR protocol could be developed to adjust the period of the above collect-and-broadcast process so that the process is executed more frequently when the status of the network changes rapidly, and less frequently when the network is stable.

Bibliography

[1] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[2] 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.

[3] A. Ballardie, P. Francis, and J. Crowcroft, "Core based trees," in *Proceedings of the ACM SIGCOMM '93*, (San Francisco, CA), September 1993.

[4] A. Ballardie, "Core based trees (CBT version 2) multicast routing." Internet RFC 2189, September 1997.

[5] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol." Internet RFC 1075, November 1988.

[6] J. Moy, "Multicast extensions to OSPF." Internet RFC 1584, March 1994.

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

[8] D. W. Wall, *Mechanisms for Broadcast and Selective Broadcast.* PhD thesis, Stanford University, June 1980.

[9] Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves, "A source-based algorithm for delay-constrained minimum-cost multicasting," in *Proceedings of the IEEE INFOCOM '95*, pp. 377–385, 1995.

[10] F. Bauer and A. Varma, "Degree-constrained multicasting in point-to-point networks," in *Proceedings of the IEEE INFOCOM '95*, pp. 369–376, 1995.

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

[12] 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.

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

[14] W. J. Clark, "Multipoint multimedia conferencing," *IEEE Communications Magazine*, May 1992.

[15] S. R. Ahuja and J. R. Esnor, "Co-ordination and control of multimedia conferencing," *IEEE Communications Magazine*, May 1992.

[16] J. Udell, "Computer telephony," *Byte*, vol. 19, no. 07, pp. 80–99, 1994.

[17] J. Oikarinen and D. Reed, "Internet relay chat protocol." Internet RFC 1459, May 1993.

[18] W. Reinhard, J. Schweitzer, G. Vlksen, and M. Weber, "CSCW tools: Concepts and architectures," *IEEE-Computer*, May 1994.

[19] M. Harrick, P. V. Rangan, and M. Chen, "System support for computer mediated multimedia collaborations," in *Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work (CSCW '92)*, pp. 203–209, November 1992.

[20] J. M. Pullen, M. Myjak, and C. Bouwens, "Limitations of Internet protocol suite for distributed simulation in the large multicast environment." Internet draft draft-pullen-lame-00.txt, September 1996.

[21] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," in *Proceedings of SIGCOMM '95*, (Cambridge, MA USA), pp. 328–341, 1995.

[22] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: A highly available file system for a distributed workstation environment," *IEEE Transactions on Computers*, vol. 39, April 1990.

[23] J. Postel and J. Reynolds, "File transfer protocol (FTP)." Internet RFC 959, October 1985.

[24] T. Berners-Lee, "Hypertext transfer protocol (HTTP)." available at ftp://info.cern.ch/pub/www/doc/http-spec.txt.Z, November 1993.

[25] T. Berners-Lee and D. Connolly, "Hypertext markup language 2.0." Internet RFC 1866, November 1995.

[26] J. R. Cooperstock and S. Kotsopoulos, "Why use a fishing line when you have a net? an adaptive multicast data distribution protocol," in *Proceedings of USENIX Technical Conference '96*, 1996.

[27] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and applications level framing," in *Proceedings of SIGCOMM '95*, (Cambridge, MA USA), pp. 342–356, 1995.

[28] M. Hofmann, T. Braun, and G. Carle, "Multicast communication in large scale networks," in *Proceedings of Third IEEE Workshop on High Performance Communication Subsystems (HPCS)*, (Mystic, Connecticut USA), August 1995.

[29] J. C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *Proceedings of IEEE INFOCOM '96*, March 1996.

[30] ATM Forum, *ATM User-Network Interface (UNI) Specification Version 3.1.* Prentice Hall, September 1994.

[31] P. Winter, "Steiner problem in networks: a survey," *Networks*, pp. 129–167, 1987.

[32] A. J. Ballardie, *A New Approach to Multicast Communication in a Datagram Internetwork.* Ph.D. thesis, Department of Computer Science, University College London, May 1995. Available via anonymous ftp from `cs.ucl.ac.uk:darpa/IDMR/ballardie-thesis.ps.Z`.

[33] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast sparse mode (PIM-SM): Protocol specifications." Internet RFC 2117, June 1997.

[34] M. Imase and B. M. Waxman, "Dynamic Steiner tree problem," *SIAM Journal on Discrete Mathematics*, vol. 4, pp. 369–384, August 1991.

[35] B. M. Waxman, "Performance evaluation of multipoint routing algorithms," in *Proceedings of INFOCOM' 93*, 1993.

[36] A. Thyagarajan and S. Deering, "Hierarchical distance-vector multicast routing for the Mbone," in *Proceedings of ACM SIGCOMM*, (Cambridge, Massachusetts), August 1995.

[37] A. Ballardie, "Core based trees (CBT) multicast routing architecture." Internet RFC 2189, September 1997.

[38] C. Shields and J. J. Garcia-Luna-Aceves, "The ordered core based tree protocol," in *Proceedings of IEEE INFOCOMM*, (Kobe, Japan), April 1997.

[39] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP architecture for inter-domain multicast routing," to appear in Proceedings of ACM SIGCOMM, (Vancouver, Canada) August, 1998.

[40] Fore Systems, Inc., *ForeRunner SBA-200 ATM SBus Adapter User Manual*, 1993.

[41] D. Dykeman, H. L. Truong, and H. J. Sandick, "Alternatives for the support of the ATM group services." ATM Forum internal contribution 95-0438, April 1995.

[42] F. Liaw, "A straw man proposal for ATM group multicast routing and signaling protocol: Architecture overview." ATM Forum internal contribution 94-0995, November 1994.

[43] R. Perlman, "Fault-tolerant broadcast of routing information," in *Proceedings of IEEE Infocom '83*, (San Diego), 1983.

[44] D. E. Corporation, "Information processing systems – data communications – intermediate system to intermediate system intra- domain routing protocol," October 1987. Also available as Internet RFC 1142.

[45] D. Bertsekas and R. Gallager, *Data Networks.* Prentice-Hall, 1987.

[46] 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.

[47] E. Fleury, Y. Huang, and P. K. McKinley, "On the performance and feasibility of multicast core selection heuristics," Tech. Rep. MSU-CPS-97-42, Department of Computer Science, Michigan State University, East Lansing, Michigan, October 1997.

[48] H. D. Schwetman, "CSIM: A C-based, process-oriented simulation language," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corporation, 1985.

[49] FORE Systems, Inc., *SPANS NNI: Simple Protocol for ATM Network Signaling (Network-to-Network Interface) Release 3.0*, 1993. available at ftp://ftp.fore.com/pub/docs/spans/spans3nni.ps.

[50] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A user-level network interface for parallel and distributed computing," in *Proc. of the 15th ACM Symposium on Operating Systems Principles*, (Copper Mountain, Colorado), pp. 40–53, December 1995.

[51] D. Johnson, D. Lilja, and J. Riedl, "A circulating active barrier synchronization mechanism," in *Proceedings of the 1995 International Conference on Parallel Processing*, vol. I, pp. 202–209, August 1995.

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

[53] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos, "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM*, vol. 39, no. 4, 1996.

[54] I. Cidon, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M. Sidi, "The OpeNet architecture," Tech. Rep. 95-37, Sun Microsystems, December 1995.

[55] I. Cidon, A. Gupta, T. Hsiao, A. Khamisy, A. Parekh, R. Rom, and M. Sidi, "OPENET: An open and efficient control platform for ATM networks," in *Proc. INFOCOM'98*, (San Francisco, CA), pp. 824–831, March 1998.

[56] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. 41 Madison Avenue, New York, N.Y. 10010: W. H. Freeman and Company, 1979.

[57] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proceedings of IEEE INFOCOM '96*, (San Francisco, California), March 1996.

[58] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the Internet." Internet draft draft-ietf-qosr-framework-00.txt, March 1996.

[59] F. Bauer and A. Varma, "ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm," in *Proceedings of IEEE Infocom '96*, (San Francisco, California), pp. 361–368, March 1996.

[60] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[61] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558–565, July 1978.

[62] H. D. Schwetman, "CSIM: A C-based, process-oriented simulation language," Tech. Rep. PP-080-85, Microelectronics and Computer Technology Corporation, 1985.

[63] D. Menasce, R. Muntz, and J. Popek, "A locking protocol for resource coordination in distributed databases," *ACM TODS*, pp. 103–138, 1980.

[64] K. Birman, "Implementing fault tolerant distributed objects," *IEEE Transaction on Software Engineering*, pp. 502–508, 1985.

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

[66] S. Singh and J. Kurose, "Electing 'good' leaders," *Journal of Parallel and Distributed Computing*, vol. 21, pp. 184–201, May 1994.

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

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

[69] M. Handley and V. Jacobson, "SDP: Session description protocol." Internet draft draft-ietf-mmusic-sdp-03.txt, March 1997.

[70] M. J. Donahoo and E. W. Zegura, "Core migration for dynamic multicast routing," in *Proceedings of IEEE ICCCN '96*, (Rockville, Maryland), October 1996.

[71] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Shauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation," in *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*, (San Diego, California), pp. 1–12, Association for Computing Machinery, May 1993.

[72] A. Gopal, I. Gopal, and S. Kutten, "Broadcast in fast networks," in *Proc. INFOCOM'90*, (San Francisco, CA), June 1990.

[73] E. Basturk and P. Stirpe, "A hybrid spanning tree for efficient topology distribution in PNNI," Tech. Rep. Research Report RC 20922, IBM Research Division, July 1997.

[74] B. Rajagopalan, "Efficient link state routing." NEC Technical Report, 1997.

[75] I. Cidon, I. Gopal, M. Kaplan, and S. Kutten, "A distributed control architecture of high-speed networks," *IEEE Trans. Commun.*, vol. 43, no. 5, pp. 1950–1960, 1995.

[76] E. C. Rosen, "Vulnerabilities of network control protocols: An example," in *SIGCOMM Computer Communications Review*, pp. 10–16, July 1981. (also published as RFC 789).

[77] Y. Huang and P. K. McKinley, "Group leader election under link-state routing," in *Proceedings of International Conference on Network Protocols, 1997.*, (Atlanta, Geogia), October 1997.