Multicast Group Membership Management

Joshua Auerbach, *Member, IEEE*, Madan Gopal, Marc Kaplan, *Member, IEEE*, and Shay Kutten, *Senior Member, IEEE*

***Abstract*—Multicast services, assisted by special hardware, are being considered as a part of high-speed wide-area networks (WANs) in order to support new generations of multiuser applications. This paper describes an application multicast service for high-speed WANs which is capable of exploiting multicast hardware. Indeed, this research was conducted in context of the spanning tree hardware structure of PARIS and of plaNET, the pioneering broadband experimental networks that predated ATM. The results of this research were also included in IBM's ATM, called Networking BroadBand Services (NBBS).**

We achieve modularity and low cost by assigning to distinct components the separate problems of: 1) naming groups; 2) finding group members in a network; 3) configuring multicast hardware; and 4) delivering multicast messages in sequence. This modularity enables, for example, both the multicast to a group to which the user initiates the joining (formed by using 1 and 2 above), on one hand, and to groups computed by the source on the other hand. We give the overall organization of our service and then describe in detail the methods used to solve the first two of the subproblems.

***Index Terms*—Broadband, broadcast, distributed algorithms, distributed control, group communication, multicast, protocols.**

I. INTRODUCTION

MULTICASTING allows one entity to communicate efficiently with multiple entities residing in a subset of the nodes in a network. Although multicasting can be simulated using multiple point-to-point messages, mechanisms that enable multidestination delivery using a single group address can provide transparency, greater efficiency, and concurrency. Many applications are natural candidates for using multicast, e.g., distributed databases [1], replicated file systems [2], resource allocation in distributed systems [3], distributed process management [4], distributed games [5], replicated procedure calls [6], and teleconferencing [7].

Most existing multicast services (and much of the research) has concentrated on local-area networks (LANs) [1], [4], [8]–[10] since these provide a broadcast medium, simplifying implementation of multicast. Multicasting is less popular on wide-area networks (WANs) because it is difficult to implement efficiency on a point-to-point medium [11]. However, the trend toward very high-speed WANs has led to renewed interest in WAN multicast services.

High-speed WANs will enable new multiuser applications (e.g., video conferencing, multimedia databases, and code distribution) which fit the multicast model well. In addition, many network control algorithms (e.g., topology maintenance, directory, and clock synchronization) used in high-speed networks can make use of multicasting to increase efficiency. These newer multicast applications have stringent requirements on bandwidth and latency which precludes simulation using point-to-point mechanisms. Consequently, the switches used in high-speed networks increasingly employ specialized hardware [12]–[17] which allows multicast (as well as point-to-point) messages to be routed through a switch with minimal overhead. Indeed, this research was conducted in context of the spanning tree hardware structure of PARIS (the world's first integrated packetized network to carry data, voice and video, and the first spatial reuse optical ring) and of plaNET, the pioneering broadband experimental networks that predated ATM. The results of this research were also included in IBM's ATM, called Networking BroadBand Services (NBBS). (See also U.S. Patent 5 634 011 [31].) Today, switches with such a specialized hardware are sold by many vendors.

II. MULTICAST MODEL OVERVIEW

In explaining how we look at the problem of building multicast services, we begin with a few definitions.

A **transport user process** (TUP) is any process which makes use of a transport service, including the multicast service. A **transport address** is used to refer to a TUP and can be used to locate it in the network. A **multicast source** is a TUP which is the source of multicast traffic. A **multicast destination** is an individual TUP which is the recipient of multicast traffic. The terms **source** and **destination** are always in terms of a particular **multicast sequence**. A **multicast sequence** is a set of *related* messages which are multicast from a single source to a set of destinations. It can be as short as one message, but it may have arbitrarily many messages and also arbitrary duration in time. It is up to higher layer protocols and applications to map their constructs (such as files, sessions, and transactions) onto single or multiple multicast sequences. A TUP which is the source of some multicast sequences may be the destination of others.

The **destination set** of a multicast sequence is the set of TUPs which may receive messages in the sequence. The membership in the destination set remains stable during a multicast sequence in the absence of failures. In the presence of failures, the destination set can lose members but cannot gain them.

Manuscript received September 22, 1994; revised September 15, 1995; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor D. Estrin. Note from the Editor-in-Chief: This paper was originally submitted in 1994 and accepted in 1996. Because of some miscommunication among the various parties concerned with getting the accepted version to publication, the paper was never published. This problem was only discovered recently. Because the paper addresses fundamental issues in the management of multicast groups, and after some deliberation, I decided to publish a slightly updated version of the paper along with this note explaining the situation.

J. Auerbach and M. Kaplan are with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: josh@us.ibm.com; makaplan@us.ibm.com).

M. Gopal is with Cisco Systems, San Jose, CA 95134 USA (e-mail: pmgopal@cisco.com).

S. Kutten is with the Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: kutten@ie.technion.ac.il).

Digital Object Identifier 10.1109/TNET.2002.808411

A **distribution vehicle** is a collection of hardware settings in the network which makes it efficient for some source or sources to multicast to a particular destination set. All multicast sequences are sent over a distribution vehicle and all messages of the same sequence use the same vehicle. The distribution vehicle is a "best effort" channel which can lose, duplicate, or reorder messages. It should provide a reverse channel on which replies and acknowledgments to the source can flow. Any additional guarantees are the province of the transport service or even higher layers, which are not discussed in this paper.

A **multicast set** is a *potential* destination set. It is a set of TUPs in one network partition whose transport addresses have been collected into a list. This list, when combined with network topology information, can be used to guide the construction of distribution vehicles which reach all members of the multicast set as destinations. The *sources* of multicast sequences directed at a multicast set *may or may not* be a member of the set.[1] Each multicast set is known by a group name, which, as we explain in the next definition, may or may not have existed prior to the creation of the set.

A **multicast community** is a set of human or organizational users (*not* TUPs) who have been assigned a **group name** for a prearranged purpose. Some of these users *may* be represented by one or more TUPs in the network; normally, those TUPs would form a multicast set whose group name is that of the corresponding multicast community. However, if the network is partitioned, there may be temporarily more than one multicast set of a given name (one in each partition).

Two key distinctions in our model set it apart from other approaches.

First, we distinguish between a **multicast community**, made up of users *outside* the network, and a **multicast set**, made up of TUPs *inside* the network. The concepts are linked in that a TUP can *represent* a human user and so a multicast set can represent a multicast community. But, for many applications, the lifetime of a group name is much longer than the lifetime of TUPs and the mechanisms needed to administer names should not depend on the distributed protocols needed to manage the more topology-sensitive aspects of multicasting.

Second, we distinguish between multicast sets, multicast sequences, and distribution vehicles. A multicast set is simply a list of TUP transport addresses representing the eligible receivers of multicasts to a particular group name. While a distributed protocol is needed to collect this list, once it has been collected it can persist across multiple multicast sequences, from the same or different sources. Any number of multicast sequences to the same set may be in progress at the same time.

Distribution vehicles will typically be created with knowledge of the present set membership and also of the network topology. The creation and maintenance of distribution vehicles is based entirely on performance considerations and is transparent to users of the service. There may be individual distribution vehicles for individual multicast sequences (depending on bandwidth requirements, for example). There may be a standing distribution vehicle for certain multicast sets. There may be certain "large" distribution vehicles shared by many multicast sets and also used for broadcast purposes.

[1]In other words, we advocate implementing only *open* groups as in the V kernel [4]. The "closed" property for certain groups can be accomplished via screening at the transport level.

As a consequence of these key distinctions, our service divides itself into four key components.

1) **Group name administration** is responsible for determining the assignment of group names to multicast sets and, where the concept is applicable, to multicast communities. Three distinct authorization paradigms are supported, as described in a later section.
2) **Set management** is responsible for maintaining dynamic lists containing transport addresses of TUPs for each multicast set.
3) **Delivery vehicle management** is responsible for creating delivery vehicles as needed, using information from set management and topology information from other network control sources.
4) **Multicast transport services** is responsible for determining the behavior of multicast sequences. If the service requirement is stronger than that provided by the distribution vehicle (e.g., reliable delivery, in-order delivery), transport protocols are used to achieve the stronger requirement.

The first two of these, collectively called **group membership management**, will be the focus of the remainder of this paper. The other two components were addressed in various papers concerning NBBS, PARIS, and plaNET (see, e.g., [16], [17], and [27]).

III. Group Name Administration

Group name administration is ultimately responsible for assigning group names to multicast sets, but, often, this assignment is indirect: First, a group name is assigned to a multicast community; later, as members of the community come to be represented by TUPs in the network, these TUPs will join or leave the multicast set of the same name. An administrative structure (and perhaps a database) is needed to administer names, but no explicit distributed algorithms. We stress here the separation between the set administration and the group name administration. This enables a simple and flexible naming structure, which can be useful for different applications that each give its own meanings to part of the names. It also enables to rid the designer of the set membership protocols from issues that actually belong to group management.

We have identified three distinct kinds of names which appear to be needed by different application contexts.

A **public name** is assigned to a multicast community which may be indefinitely large and includes all users who know the group's name. There is no formal process for joining a public multicast community. No authentication is associated with public names, but they are still administered to prevent accidental collisions.

A **restricted name** is associated with a multicast community whose membership is controlled and can be checked by an *authentication process* (see, e.g., [19]). An authentication process is consulted whenever a TUP attempts to join a multicast set with a restricted name. The TUP must represent an authorized user or it will be rejected; how the representation relationship is proven depends on the particular authentication mechanism. Authentication processes are administered to assure that each is responsible for a disjoint part of the group name space. Each authentication process is then responsible for assigning its own names and for authenticating their users.

Although public and restricted names differ on the authentication dimension, both represent multicast communities which exist apart from the network. Both can support *stable* communities, such as parties interested in particular documents or a replicated network service. TUPs representing community members can appear and disappear in different parts of the network (including different partitions) at arbitrary times. In order to tolerate this nondeterminism, the style of set management associated with these types of groups must be *distributed* and includes the basic operations `join_set` and `leave_set`.

Transient names are not associated with pre-existing communities and are not "administered." They are assigned directly to sets of TUPs. Our goal here is to streamline administration when a long-lasting name is not needed and when the TUPs already know each others' transport addresses as a result of prior application-level exchanges. Conference calls, either traditional or multimedia, are expected to use this mechanism. Collisions are avoided by generating group names algorithmically. One set member appends a suffix (e.g., a time stamp) to its transport address. This TUP becomes the "set leader" (see Section IV) and also controls the set membership. The associated style of set management is thus *centralized* and includes the basic operations `create_set`, `destroy_set`, `add_to_set`, and `delete_from_set`.

Although we distinguish between a distributed and centralized style of set management, all other operations of the multicast service (those relating to distribution vehicles and multicast sequences) behave identically regardless of which style of set management is employed. Consequently, we need a uniform syntax for group names, which includes but distinguishes all three types. This syntax is used internally; human users may use "friendlier" names, if desired, via a directory service.

We assume that group names consist (mostly) of printable characters and that there are three reserved characters, to which we assign the terminal names ⟨Restrict⟩ (for the restricted groups), ⟨Separator⟩, and ⟨Transient⟩ (for the transient groups).

Two other terminals are the following.

- **Name-segment**, an arbitrary string of characters not including any of the reserved ones;
- **Trans-stem**, a valid transport address, which may not include the ⟨Transient⟩ character.

In the syntax below, we see that multiple Name-segments can be used as a part of the name of a group. This caters to hierarchical schemes and can be used also for multiple protocols, where each looks at a different part of the name. The Trans-stem terminal identifies the creating node of a transient group. A possible syntax for group names is as follows:

```
groupname ::= Struct-name
 ::= Struct-name ⟨Restrict⟩ Struct-name
 ::= Trans-stem ⟨Transient⟩ Struct-name
Struct-name ::= Name-segment ::= Name-segment
 ⟨Separator⟩ Struct-name
```

The ⟨Transient⟩ and ⟨Restrict⟩ delimiters mark the name as belonging to a transient group or a restricted group, respectively. A name containing neither of those delimiters is assumed to be public. The ⟨Transient⟩ delimiter, furthermore, permits parsing to find the transport address of the owning TUP (Trans-stem). Recall that for restricted groups, an authentication authority is to be consulted when an agent declares that it belongs to such a group. The ⟨Restrict⟩ delimiter defines the Struct-name to the left as the name of the authentication process to be consulted. Otherwise, Struct-names (arbitrarily segmented names) are used both in naming authentication processes and in segmenting public names. The power to establish authentication processes and to assign names can thus be delegated through arbitrary levels of hierarchy.

IV. Set Management

Set management is a necessary component of any multicast service organized as we have proposed. It manages membership information for the three types of group names. As described in the previous section, public and restricted names allow a TUP to issue the operations `join_set` and `leave_set`, while transient names permit `create_set`, `destroy_set`, `add_to_set`, and `delete_from_set` operations to be issued by the owner of the set and `leave_set` operation by a member of the set.

A key feature of our approach is to ensure that any source can easily find out the set membership by issuing a `query_set` operation anytime. We believe that in WANs, it is crucial to be able to get this information in order to build an efficient delivery vehicle on top of the point-to-point communication medium.

There is at most one multicast set in each network partition with a given name. This implies that if two partitions regain connectivity to each other, sets with the same name will merge together to yield exactly one set in the resulting network. To support this requirement, set management algorithms should be able to handle changes in membership, both due to members joining or leaving (or being added or deleted) and due to the dynamic topology changes in the network. For the latter case, set management has to cooperate with the topology maintenance facilities provided by the network.[2]

Set management must be implemented in a distributed manner with different nodes performing specific functions and working together cooperatively to provide the service. Here, we propose an algorithm for accomplishing that goal.

- **Set agent**. Each node contains a process called the **agent**, which knows about and acts on behalf of the various TUPs located in that node.[3]
- **Set leader**. The set **leader** is a distinguished agent for a given group name which knows the membership of the entire set as opposed to the agent in a node which only knows

[2]We assume that functionality provided by the topology maintenance facilities includes at the very least a way for nodes to determine reachability to other nodes.

[3]We assume that it is possible to determine the set agent's transport address from the address of any TUP in the node.

about its local members. The set leader is determined dynamically for sets with public and restricted names, while for those with transient names, the agent at the node where the owner resides is the set leader.

- **Registrar**. The **registrar** is a distinguished agent with two basic functions, the first being to coordinate the leadership resolution process (thus, helping to unite sets of the same group) and the second being to direct the TUPs outside the multicast set to the set leader.

A variety of techniques could be used to unambiguously determine the registrar in a given network, depending on the type of control information available in a given network architecture. Many existing techniques [17], [20]–[22] use *link state routing* where each node broadcasts its local node and link characteristics to all the nodes in network. As a result of these topology updates, each node can discover all the nodes which are in the same network partition as itself. Thus, a simple method is to pick the agent in the node with the highest address as the registrar. Each agent can easily determine the current registrar based on its topology information.

Each agent maintains three lists for each set: MyLocalMembers, MyChildren, and MyMembers. MyLocalMembers contains the list of local TUPs in a set. MyChildren contains the list of other agents reporting to this agent and the members reported by each of them. This list is nonempty only if the agent is the leader or was a leader at some point in time. It is possible to have an empty MyLocalMembers list and nonempty MyChildren list at an agent if all the local TUPs leave the set, voluntarily or involuntarily. MyMembers list at the leader contains the entire set membership, while at other agents it contains partial membership information.

In addition, the registrar maintains the SetLeaders list, which contains the addresses of the current leaders for various group names.

A. Set Management for Public and Restricted Names

The set membership information for public and restricted names is managed in a distributed fashion. The code for the algorithm appears in Fig. 4. We now describe the algorithm informally.

Leader Resolution: Initially, every agent is a contender for the leadership. Each potential leader sends an ASSERT message to the registrar. For example, this may happen when the first TUP issues a `join_set` for the group name at that node. If the registrar has no entry about this group name in the SetLeaders list, the asserting agent becomes the leader for that set. The first `join_set` initializes the set.

On the other hand, if there is a leader already in SetLeaders and it is currently reachable, the registrar determines whether the leadership should be changed based on other criteria. In any case, the losing contender is notified of the winning leader in the reply to the ASSERT.

A simple criterion used to resolve the leadership contention is to pick the agent which asserted first. This is the criterion used in our algorithm. Other alternatives include selecting the agent with the higher node address, or the agent with larger set membership.

Control Tree: As a result of the leadership resolution process, the losing contender finds out about the current leader. The loser then becomes a *child* of the winner. This relationship is established by having a **control connection** between the two agents. At a later point in time, the *parent* may give up its leadership and establish a control connection with another agent.

In general, the control connections among agents (for a given group name) belonging to the same network partition will form a **control tree** with the current set leader being at the root of this tree and the internal nodes in the tree being agents which were leaders at some point in time in the past. A control connection between two agents is assumed to have the following properties.[4]

1) **Reliable message transmission**. The membership updates are carried on the control connections and need to be reliably transmitted.
2) **First-in-first-out (FIFO) ordering of messages**. This ensures that the updates will be received and processed at the leader in the correct sequence.
3) **Outage notification to end points**. The outage notification property serves to identify potential network failures which can partition the network and result in membership changes in the set.

Membership Updates: The membership information is conveyed from the agents to the leader on the control tree. Each agent reports its MyMembers list to its parent using an UPDATE-TO-PARENT message. The parent updates its MyChildren list as a result of this message and recomputes MyMembers using MyChildren and MyLocalMembers. It then reports its new MyMembers list to its parent. This process continues until the leader gets the information and updates its MyMembers list.

The UPDATE-TO-PARENT message is sent by an agent in response to a change in its MyMembers list (which can change due to a change in MyLocalMembers or in MyChildren), or when a new parent is acquired.

There are two important considerations which have to be addressed in order to ensure correctness of the membership information.

1) **Sequencing join/leave information**. Consider the scenario in Fig. 1 where the initial set consists of $\{A, B, C\}$. TUP D issues a `join_set` verb immediately followed by a `leave_set`. If the two corresponding updates are not received at the leader in the same order (this may happen if they are sent as datagrams), there is a potential for inconsistency. In the example, assuming that the leader treats the out-of-order update as a spurious message, the end result is that the leader's membership list will have $\{A, B, C, D\}$ rather than $\{A, B, C\}$.

   The correct ordering is enforced in our design because all updates flow between agents on control connections rather than as datagrams. The issue of control connection failure is dealt with in a later section.

[4]These properties can be accomplished by means of end-to-end protocols or be provided by the network layer.

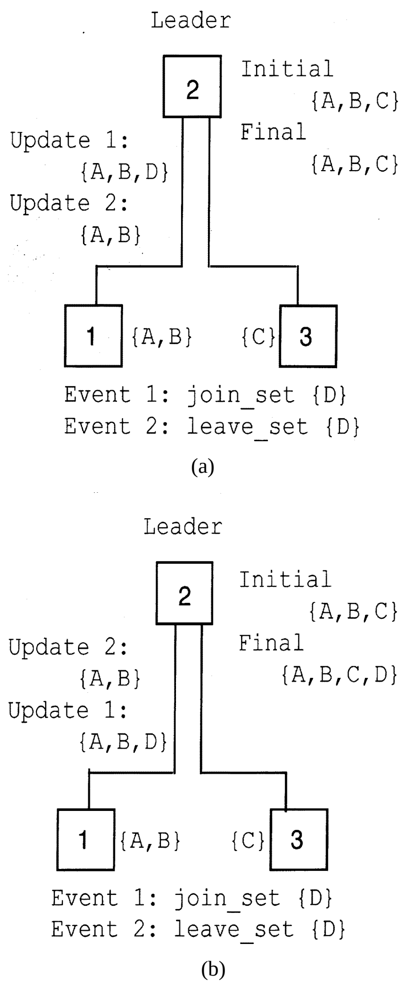


Fig. 1. Sequencing join/leave information. (a) `join_set` reaches first. (b) `leave_set` reaches first.

2) **Updating set membership information**. A related inconsistency can arise due to differences in time taken to traverse different control connections. For example, consider the scenario in Fig. 2 where the initial set is $\{A, B, C\}$ and TUP D issues a `join_set` followed by a `leave_set`. Subsequently, the link between nodes 4 and 1 fails and agent 4 becomes the child of agent 2 which is the leader.[5] At that time, suppose TUP D issues a `join_set` again.

Let us illustrate what could happen if the individual `join_sets` and `leave_sets` were to be propagated to the leader. Note that the ordering between the first `join_set` and the `leave_set` is guaranteed since they flow on the same control connection. However, the second `join_set` flows on a different connection and is not guaranteed to reach the leader after the `leave_set`.[6] Thus, if the leader received `join_set (D)`, `join_set (D)`, and then `leave_set (D)`, the final outcome depends on what action it takes upon receiving the second `join_set (D)`. For example, if the second `join_set (D)` is simply discarded as a duplicate, the end result is $\{A, B, C\}$, which is incorrect.

Our solution to this problem is for agents to always propagate their current membership list (i.e., MyMembers) to their parent and for the parent to take the *union* of its children's' membership lists and its local membership list (MyLocalMembers) as its new MyMembers list. In the example described above, agent 4 propagates $\{D\}$ to agent 1 upon receiving the verb `join_set (D)` and { } upon receiving the verb `leave_set (D)`. Agent 1 in turn, propagates $\{A, B, D\}$ (Update 1) and $\{A, B\}$ to the leader upon receiving the corresponding updates from agent 4. Agent 4 propagates $\{D\}$ (Update 3) to the leader upon receiving the second `join_set (D)`. Thus, the leader has the correct set $\{A, B, C, D\}$ after the three updates have been applied. Note that there may still be *transient* inconsistencies. In the above example, the leader's membership list always contains TUP D as a member (given that the order of updates reaching the leader is Update 1, Update 3, and Update 2), although D was not a member for the period of time between issuing the `leave_set` and the second `join_set`.

Membership Queries: Any TUP can issue a `query_set` to the local set agent for a given group name, whether or not it is a part of the destination multicast set. The agent can reach the set leader by traversing the control tree to the root. Alternatively, it can get the address of the current set leader from the registrar and forward the query to it. If the agent does not represent any TUP for that group name, it simply uses the second approach.

Notice that the membership list returned by the leader is only current as of the instant the `query_set` was received.

Registrar Changes: The registrar may change as a result of the following events.

1) The current registrar fails.
2) The current registrar is unreachable.
3) A new node with a higher address than that of the current registrar joins the network.
4) Two previously disconnected network partitions regain connectivity.

Assuming that every node locally maintains the current topology map of the network, each agent can, by monitoring the topology updates, determine the events that can trigger a registrar change.[7] In each case, the agent with the highest node address is selected as the new registrar.

In 1) and 2), in the partition which lost its registrar, all set leaders send ASSERTs to the new registrar. In the other partition (if any), the registrar does not change. The new registrar builds up its leaders list as a result of these ASSERTs.

In 3) and 4) (see Fig. 3), the registrar of one of the previous partitions becomes the registrar for the whole network after the partitions gain connectivity. Since a set leader in the other partition finds that its registrar has changed, it sends an ASSERT to its new registrar and discovers that there is already a leader for that group name. It then becomes a child of the current leader.

Control connection failure. When an agent discovers that its control connection to its parent is broken (recall that in the event

[5]As described earlier, agent 4 will send an ASSERT to the registrar and find out that agent 2 is the current leader. It will then establish a control connection with agent 2 (shown in dotted lines in Fig. 2).

[6]This is an unlikely occurrence, but a pathological scenario can be constructed where the delays on the connection between node 1 and 2 are very large compared to the connection between node 4 and 2.

[7]In other network architectures, a different mechanism could be employed to detect registrar change events depending on the functionality provided by the topology maintenance facilities.

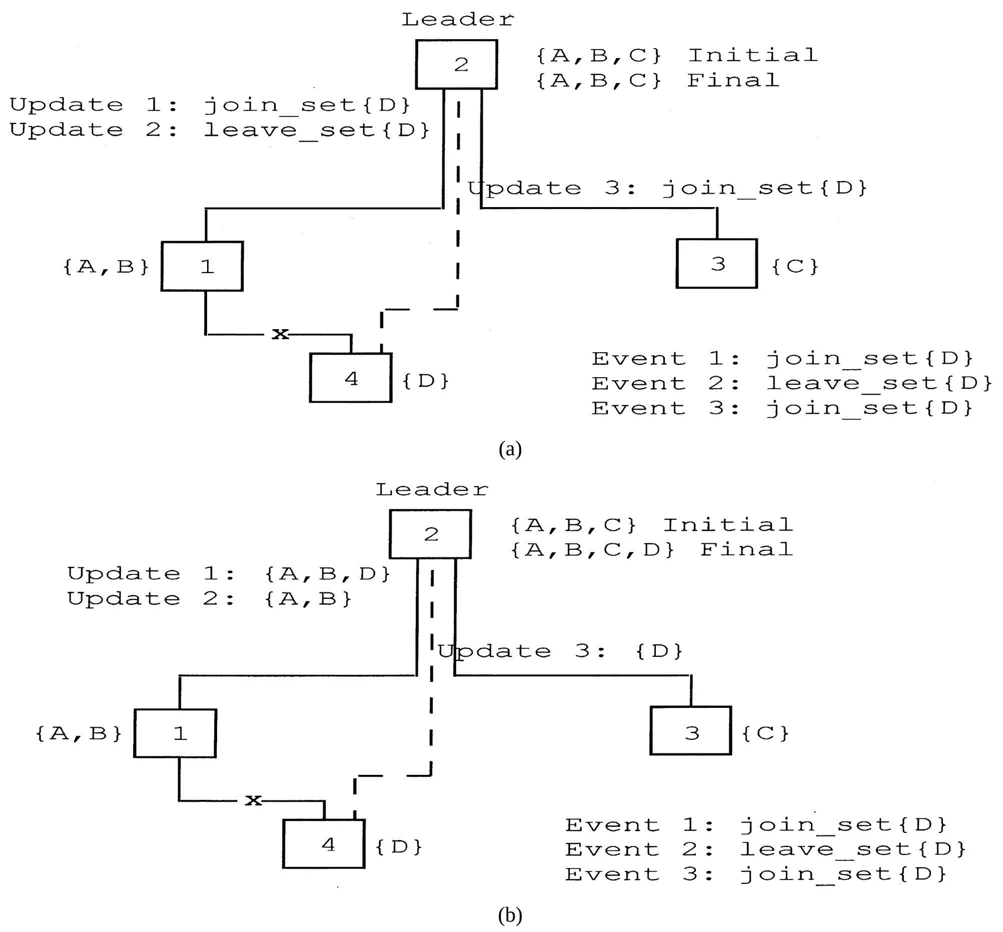


Fig. 2. Updating set membership information. (a) Updates carry `join_set/leave_set`. (b) Updates carry updated membership lists.

of an outage, both end points of a connection are notified), it retries to set up a connection to the parent, possibly on a different path. If successful, it simply sends the parent its MyMembers list again. If the parent is not reachable, the child contends for leadership by sending an ASSERT message to the registrar.

From the parent's side, a connection outage results in the parent removing all the information about the particular child from its MyChildren list and recomputing its membership. The changes are also propagated up to the leader on the control tree as described.

Correctness Properties: Several properties of the algorithm can be stated and proven in a straightforward manner based on the properties of the underlying topology maintenance and updating procedure, [17], [20]. Proofs are easy and are, therefore, omitted.

Property 1: All agents in a given network partition agree on a single registrar within a finite amount of time after topology changes in the network have ceased.

This can be concluded from the property of topology update algorithm to converge the view of the topology of all the nodes in a connected component to the correct topology if no more topology changes occur.

Property 2: The leadership resolution process results in a single set leader in a given network partition within a finite amount of time after all the ASSERTs have stopped in that partition.

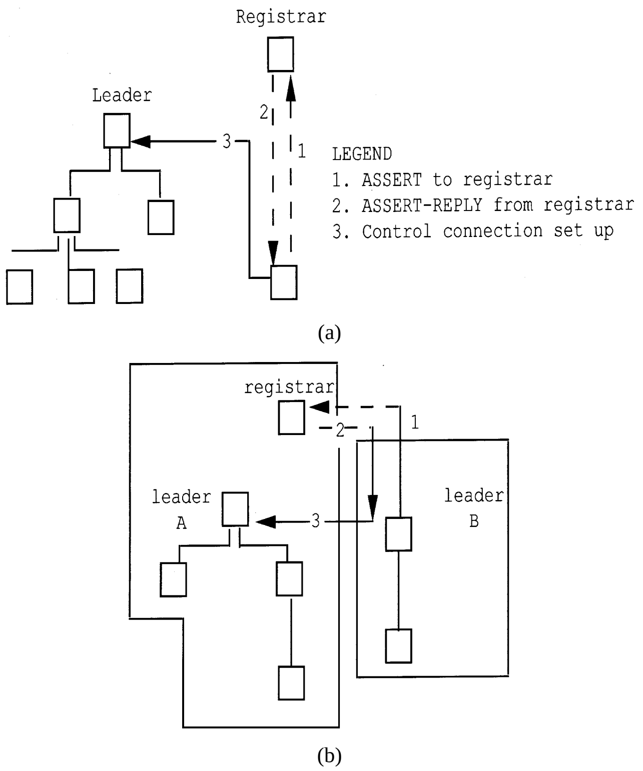


Fig. 3. Two sets regain connectivity. (a) New user issues a `join_set`. (b) Two network partitions join.

Data Structures

```
Variables at each member of each group name
      MyParent [] :  address         (* Parent agent address *)
      MyLocalMembers [] :  List of local transport user processes (TUPs)
      MyMembers [] :  List of member TUPs
      MyChildren [] :  List of <address, members>

Variables at every node
      Registrar :  address

Additional variables at the Registrar
      SetLeaders :  List of <group name, address>   (* Leader address *)
```

Algorithm

```
Starting
      MyParent = NIL
      MyLocalMembers = NIL
      MyMembers = NIL
      MyChildren = NIL

Receive JOIN-SET (GID) from TUP
      If first JOIN-SET for GID then begin
          MyParent [GID] = MyAddress   (* Initially everyone is a leader *)
          MyLocalMembers [GID] = {TUP}
          MyMembers [GID] = {TUP}
          MyChildren [GID] = NIL
          Send ASSERT (GID) to Registrar
      end
      Else begin
          MyLocalMembers [GID] = MyLocalMembers [GID] ∪ {TUP}
          MyMembers [GID] = MyLocalMembers [GID] ∪ MyMembers [GID]
          If MyParent [GID] <> MyAddress then
              Send UPDATE-TO-PARENT (GID, Members [GID]) to MyParent [GID]
      end

LEAVE-SET (GID) from TUP
      MyLocalMembers [GID] = MyLocalMembers [GID] - {TUP}
      MyMembers [GID] = MyMembers [GID] - {Tup}
      If MyMembers [GID] = NIL then begin
          If MyParent [GID] = y (* other than MyAddress*) then begin
              Terminate control connection to y
              MyParent [GID] = MyAddress
          end
      end
      Else If MyParent [GID] <> MyAddress then
              Send UPDATE-TO-PARENT (GID, MyMembers [GID]) to MyParent [GID]

QUERY-SET (GID) returns members
      Send QUERY-LEADER (GID) to Registrar
      Wait for QUERY-LEADER-REPLY (GID, Leader)
      If Leader <> ?  then begin
          Send QUERY-SET (GID) to Leader
          Wait for QUERY-SET-REPLY (GID, members)
      end
      Else error
Receive ASSERT-REPLY (GID, y) from Registrar
      If MyParent [GID] = MyAddress then begin
          Set up control connection to y
          MyParent [GID] = y
          Send UPDATE-TO-PARENT (GID, MyMembers [GID]) to y
      end

Receive UPDATE-TO-PARENT (GID, members) from x
      If <x,s> is in MyChildren [GID] then (* some value s *)
          Replace <x,s> by <x,members> in MyChildren [GID]
      Else
          Add <x,members> to MyChildren [GID]
      MyMembers [GID] = ∪ {value:  for each <child,value> in MyChildren[GID]}
      MyMembers [GID] = MyLocalMembers [GID] ∪ MyMembers [GID]
      If MyParent [GID] <> MyAddress then
          Send UPDATE-TO-PARENT (GID, MyMembers [GID]) to MyParent [GID]

Detect Control Connection Failure to z
      If MyParent [GID] = z then begin
          MyParent [GID] = MyAddress   (* I am Leader now *)
          Send ASSERT (GID) to Registrar
      end
      else If <z,s> is in MyChildren [GID] then begin
              Remove <z,s> from MyChildren [GID]
              MyMembers [GID] = ∪ {value:  for each <child,value> MyChildren [GID]}
              MyMembers [GID] = MyLocalMembers [GID] ∪ MyMembers [GID]
              If MyParent [GID] <> MyAddress then
                Send UPDATE-TO-PARENT (GID, MyMembers) to MyParent [GID]
          end
- - - - - - - - -
```

At the Registrar

```
- - - - - - - - -
Receive ASSERT (GID) from x
      If Registrar = MyAddress then begin
          If <GID, y> is in SetLeaders then begin
              If y <> x and y is still reachable from MyAddress then
                  Send ASSERT-REPLY (GID, y) to x
              Else If y <> x then   (* y is not reachable *)
                  Replace <GID, y> by <GID, x> in SetLeaders
          end
          Else add <GID, x> to SetLeaders
      End

Receive QUERY-LEADER (GID) from x
      If Registrar = MyAddress then begin
          If <GID, y> is in SetLeaders then
              Send QUERY-LEADER-REPLY (GID, y) to x
          Else Send QUERY-LEADER-REPLY (GID, ?)  to x   (* leader for GID no
      end
- - - - - - - - -
```

At Every Node

```
- - - - - - - - -
Starting
      Registrar = MyAddress
      SetLeaders = NIL

With each topology change indicating registrar change
      If Highest Address in network <> Registrar then begin
          Registrar = Highest Address in network
          SetLeaders = NIL
          For each GID such that MyParent [GID] = MyAddress do
              Send ASSERT (GID) to Registrar   (* I am a Leader *)
      end
```

Fig. 4. Set management algorithm.

Recall that the leaders look for each other at the registrar. Thus, the property follows from Property 1.

Property 3: In the absence of partitions, the membership information at the set leader is guaranteed to be correct within a finite amount of time after changes in membership have stopped.

This follows from Property 2 and from the fact that the algorithm forwards the whole membership list of a node to its parent.

In other words, we have the following property.

Property 4: Each set member exists in the membership list of the set leader within a finite amount of time after all topology changes have ceased.

B. Set Management for Transient Names

Set maintenance for transient group names is relatively simple. The agent to which the TUP issuing the `create_set` reports is the leader. Any other agent, whether it supports the members of the set or not, can determine the leader by simply parsing the group name appropriately (recall that owning TUP process identity is part of the transient-style group name) and determining the address of the corresponding set agent. It can then send a `query_set` directly to the leader and get back the membership list.

Recall that the owner of the set can modify the set using `add_to_set` and `delete_from_set` operations. In each case, the agent coresident with the owner (which is the leader of the set) simply modifies its membership list. It is also possible for members to leave the set on their own. This can be handled by maintaining control connections (and carrying updates on these connections) as for public and restricted names with the exception that the control tree has depth of 1 and the leader does not change.

For sets with transient names, only the partition containing the owner survives in the event of a partition. Thus, members not currently reachable are not part of the set.

V. Related Work

Multicast communication, though mostly on LANs, has received some attention even before the preliminary version of this paper was published [30]. We mention some later work in Section VI. In this section, we briefly describe the group membership management schemes used in four previous systems and compare these to our scheme.

Cheriton and Zwaenepoel [4] describe the management of process groups in the V-kernel. In their system, the information about the *host group* (i.e., the set of hosts on which the group members reside) is encoded in the group name. Each host maintains knowledge about the local membership in that group and about the host groups of which it is a member. It can, thus, determine whether to accept a packet and, having accepted it, deliver it to all the local processes. A key assumption in their work is that of the underlying broadcast medium provided by a LAN. Thus, the set membership need not be known explicitly for communication to occur. In WANs, it may be important to know the membership at the outset, so that an efficient delivery vehicle (e.g., tree) may be established. A scheme is proposed in [4] which consists of querying all kernel servers to determine the complete membership. However, this method may be inefficient and time consuming in WANs.

Navaratnam *et al.* [8] also describe a group communication mechanism built on top of the V-kernel. Their scheme is similar to ours, in that they have a primary manager and secondary managers to manage the group membership. However, there are two differences between their work and ours. First, both primary and secondary managers in their scheme maintain only local membership information, similar to Cheriton and Zwaenepoel. Second, they do not clearly state how a source which is not part of the group can find out information about the group membership or communicate with group members.

A different approach is taken by Birman and Joseph [1], where each group member maintains complete group membership. They support group membership management in the ISIS system using a special multicast primitive, GBCAST, which allows group modifications to be atomic and ordered with respect to all other transactions. Thus, every group member has an accurate record of the group's membership and status. This is useful for fault-tolerant distributed applications. However, we believe that many applications do not require this level of global agreement and can do without the associated cost of providing this service. Their system also does not address the case where the source is not a part of the destination set.

Deering [23] describes extensions to a host implementation of the Internet Protocol (IP) to support an internetwork multicast datagram service. In this proposal, there is at least one *multicast agent* directly attached to every IP network. Multicast agents maintain the local host membership table, the host groups to networks mapping, and the routing table. Hosts on a network communicate with a multicast agent on that network to create new groups or to modify membership of existing groups. The multicast datagram is routed by mapping the IP host group addresses to a set of local network addresses, which may be a local multicast address, a general broadcast address, or a list of local unicast addresses. This works well for a multicast datagram service but does not address the requirements of high-bandwidth traffic, e.g., video, which mandates establishment of an efficient delivery vehicle prior to communication. In contrast, our model of treating group membership management and delivery management as orthogonal issues can lead to a more modular and efficient implementation.

VI. Conclusion

The clear delineation of the responsibilities of set management from the other components sets this approach apart from the others reviewed in the previous section.

Since the preliminary version of this paper was published [30], a lot of work has been performed in the area of multicasting in broadband WANs. Surveying all this work would be beyond the scope of this paper. The reader can look, e.g., at PNNI [25] in the context of ATM, and the interdomain multicast routing (IDMR) working group of the IETF, in the context of the Internet [24]. Indeed, ideas suggested in this research are employed in current protocols. In particular, a local hardware multicasting facility is available in today's switches, and this hardware in a collection of nodes can be set up so as to serve as a delivery vehicle. Such a function to construct a nonlocal delivery vehicle out of local facilities was assumed in this paper and implemented originally in PARIS and plaNET experimental networks, as well as IBM NBBS. Moreover, in today's designs, the set management creation and maintenance indeed are separated from the setup and maintenance of the delivery vehicle.

More broadly, we have outlined a methodology for multicasting in high-speed WANs which has the necessary flexibility to support a wide range of applications. Modularity is a goal throughout. Each component has a simple task, and complex interdependencies between components are avoided. Within the single component, called set management, the functions assigned to the *registrar* and those assigned to set *agents* and set *leaders* are clearly distinguished. It would be possible, for example, to change the method of determining the registrar without changing other aspects of the algorithm. This modularity enables, for example, both the multicast to a group to which the user initiate the joining (formed by using 1 and 2 above), on one hand, and to groups computed by the source, on the other hand. This approach too became rather standard in later designs.

The service implements multicasting of the "open" sort as defined in [4]. "Closed" behavior can be obtained by arranging at the transport level to reject connections from outside the group.

The basic construct of the service is a multicast *sequence*, which is a set of related messages from one source to one destination set. Transport services may be employed optionally to strengthen the reliability within a single sequence, but it is up to still higher layers to deal with the correlation of multiple sequences.

To assure that optimal delivery vehicles can be constructed, the set management component maintains distributed and (asymptotically) accurate lists of those members of each group which are present in each partition of the network. To assure that the names given to multicast groups do not collide and that various needed styles of authentication can be supported, a group name administration component defines three types of names (public, restricted, and transient).

References

[1] K. Birman and T. Joseph, “Reliable communication in the presence of failures,” *ACM Trans. Comput. Syst.*, vol. 5, no. 1, pp. 47–76, Feb. 1987.

[2] K. Marzullo and F. Schmuck, “Supplying high availability with a standard network file system,” in *Proc. 8th Int. Conf. Distributed Computing Systems*, 1988, pp. 447–453.

[3] D. J. Farber, J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe, “The distributed computing system,” in *Proc. IEEE COMPCON*, 1973, pp. 31–34.

[4] D. Cheriton and W. Zwaenepoel, “Distributed process groups in the V-kernel,” *ACM Trans. Comput. Syst.*, vol. 3, no. 2, pp. 77–107, May 1985.

[5] E. J. Berglund and D. Cheriton, “Amaze: A multiplayer computer game,” *IEEE Software*, vol. 2, pp. 30–39, May 1985.

[6] E. C. Cooper, “Circus: A replicated procedure call facility,” in *Proc. 4th Symp. Reliability in Distributed Software and Database Systems*, Silver Spring, MD, Oct. 1984, pp. 11–24.

[7] L. Aguilar, J. Garcia-Luna-Aceves, D. Moran, E. Craighill, and R. Brungardt, “Architecture of a multimedia teleconferencing system,” in *Proc. ACM SIGCOMM*, 1986, pp. 126–135.

[8] S. Navaratnam, S. Chanson, and G. Neufeld, “Reliable group communication in distributed systems,” in *Proc. 8th IEEE Int. Conf. Distributed Computing Systems*, San Jose, CA, 1988, pp. 428–437.

[9] S. E. Deering, “Multicast routing in internetworks and extended LANs,” *Comput. Commun. Rev.*, vol. 18, no. 4, pp. 55–64, 1988.

[10] M. Ahamad, M. H. Ammmar, J. M. Bernabeu-Auban, and M. Khalidi, “Using multicast communication to locate resources in LAN-based distributed system,” in *Proc. 13th Conf. Local Computer Networks*, Minneapolis, MN, Oct. 1998, pp. 193–202.

[11] K. Paliwoda, “Transactions involving multicast,” *Comput. Commun. Rev.*, vol. 11, no. 6, pp. 313–318, Dec. 1988.

[12] J. S. Turner, “Design of a broadcast packet network,” in *Proc. INFOCOM*, 1986, pp. 667–675.

[13] I. Gopal and I. Cidon, “PARIS: An approach to private integrated networks,” *J. Analog Digital Cabled Syst.*, vol. 1, pp. 77–85, June 1998.

[14] E. Arthurs, R. Boorstyn, and T. T. Lee, “The architecture of a multicast broadband packet switch,” in *Proc. INFOCOM*, 1988, pp. 1–8.

[15] K. Y. Eng, M. G. Hluchyj, and Y. S. Yeh, “Multicast and broadcast services in a knockout packet switch,” in *Proc. INFOCOM*, 1988, pp. 29–34.

[16] I. Cidon, I. Gopal, and S. Kutten, “New models and algorithms for future networks,” in *Proc. ACM PODC*, Toronto, ON, Canada, 1988, pp. 75–89.

[17] I. Cidon, I. Gopal, S. Kutten, and M. Kaplan, “A distributed control architecture of high-speed networks,” *IEEE Trans. Commun.*, vol. 43, pp. 1950–1960, May 1995.

[18] I. Gopal, C. M. Gopal, R. Guerin, R. Janiello, and M. Kaplan, “The plaNET/ORBIT high-speed network,” *J. High Speed Networks*, vol. 2, no. 3, pp. 1–38, 1993.

[19] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, “The Kripto–Knight family of light-weight protocols for authentication and key distribution,” *IEEE/ACM Trans. Networking*, vol. 3, pp. 31–41, Feb. 1995.

[20] A. E. Baratz, J. P. Gray, P. E. Green Jr., J. M. Jaffe, and D. P. Pozefsky, “SNA networks of small systems,” *IEEE J. Select. Areas Commun.*, vol. SAC-3, pp. 416–426, May 1985.

[21] J. M. McQuillan, I. Richer, and E. C. Rosen, “The new routing algorithm for ARPANET,” *IEEE Trans. Commun*, vol. COM-28, pp. 711–719, May 1980.

[22] J. Rekhter, “The NSFNET backbone SPF-based interior gateway protocol,” Network Working Group, RFC 1074, 1988.

[23] S. E. Deering, “Host extensions for IP multicasting,” Network Working Group, RFC 988, 1986.

[24] Inter-domain multicast routing (IDMR). [Online]. Available: http://www.ietf.org/html.charters/idmr-charter.html

[25] ATM Forum, Private network-node interface (PNNI) specifications. [Online]. Available: http://www-comm.itsi.disa.mil/atmf/pnni.html

[26] N. Budhiraja, M. Gopal, M. Gupta, E. A. Hervatic, S. J. Nadas, and P. A. Stirpe, “Multicast network connection architecture,” *IBM Syst. J.*, vol. 34, pp. 590–603, 1995.

[27] G. A. Marin, C. P. Immanuel, P. F. Chimento Jr., and I. S. Gopal, “Overview of the NBBS architecture,” *IBM Syst. J.*, vol. 34, pp. 564–589, 1995.

[28] M. Peyravian, R. A. Bodner, C.-S. Chow, and M. A. Kaplan, “Efficient transport and distribution of network control information in NBBS,” *IBM Syst. J.*, vol. 34, pp. 640–658, 1995.

[29] G. Lebizay, C. Galand, D. Chevalier, and F. Barre, “A high-performance transport network platform,” *IBM Syst. J.*, vol. 34, pp. 705–724, 1995.

[30] J. Auerbach, P. M. Gopal, M. Kaplan, and S. Kutten, “Multicast group membership management in high-speed wide area networks,” in *Proc. 11th Int. Conf. Distributed Computing Systems*, Arlington, TX, May 1991, pp. 231–238.

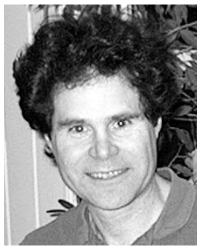
[31] J. S. Auerbach *et al.*, “ Distributed management communications network,” U.S. Patent 5 634 011, May 27, 1997.

Joshua Auerbach (M’89) received the Ph.D. degree from Yale University, New Haven, CT.

He has been a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY, since 1983, specializing in distributed middleware systems and integration of heterogeneous software elements. He contributed to the recently announced Websphere MQ Event Broker product from IBM.

Madan Gopal received the Ph.D. degree in computer science from the University of Waterloo, Waterloo, ON, Canada.

From 1985 to 1995, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, as a Research Staff Member. During this time, he worked on algorithms and protocols for network control in the area of directory and multicast services. He was one of the key contributors to IBM’s Broadband Networking Services Architecture (NBBS). He joined Cisco Systems, San Jose, CA, in 1995 to head the software development effort for the LS1010 ATM Switch and later for the Catalyst 8500 family of products. Currently, he is Director of Engineering in the Metro Services Business unit working on the ONS 155xx family of Metro DWDM products.



Marc Kaplan (M’87) received the Ph.D. degree from Princeton University, Princeton, NJ, in electrical engineering and computer science in 1978. His dissertation work was the development of novel techniques for global analysis of type information within programs with applications to compiler optimizations.

He is currently a Research Staff Member with the IBM T. J. Watson Research Center, Yorktown Heights, NY, working the Gryphon Distributed Publish/Subscribe Messaging Project. His contributions to Gryphon include architectures and protocols for security and for highly scalable and robust multicast. He has held various staff and management positions at IBM since the summer of 1978. He has designed and programmed file and security subsystems, operating system kernels, communications, and security systems. In the late 1980s and early 1990s, he architected, coded, and managed the software for the Paris/2-Planet-Orbit Gigabit networking project. It was during this period, when wide-area gigabit networking was still exotic technology, that he collaborated with his coauthors of this paper. Some of this technology was subsequently integrated into ATM and Internet protocols and products. His later research and development work resulted in IBM’s patented Cryptolope™ Technology, which is now a feature of the IBM SecureWay™ and the IBM Digital Library.

Shay Kutten (SM'96) received the M.S. degree in computer science for work on scheduling of radio broadcasts and the Ph.D. degree in computer science for work on distributed network algorithms from the Technion—Israel Institute of Technology, Haifa, Israel, in 1984 and 1987, respectively.

From 1987 to 1996, he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, as a Postdoctoral Fellow, as a Project Leader, as the Manager of the Network Architecture and Algorithms group, and as a Research Staff Member. He led the network security project which developed the security architecture for several IBM products, and developed algorithms for network control, security, and distributed processing control, which were later used in IBM's products. He has contributed to the theory of distributed computing, mostly by introducing new theoretical subjects, many of them inspired by his work on practical issues, and by giving well-founded solutions to practical problems. Currently, he is the Head of the Information Systems area of the Davidson Faculty of Industrial Engineering and Management at the Technion. He is an Area Editor (for security, reliability, and availability) of the ACM *Journal on Selected Topics in Mobile Networks and Applications*, a Member of the Editorial Board of *Computer Networks*, and was a Member of the Editorial Board of the ACM *Wireless Networks*. He has served on program committees for several conferences and workshops, and was the Chairman of the Program Committee of the 1998 DISC Conference.

Prof. Kutten received the IBM Corporation Outstanding Innovation Award (OIA) for his work on distributed control protocols that were basis to the distributed control of IBM Broadband Networking Services architecture, (NBBS) in 1993. He received the IBM Outstanding Innovation Award (OIA) for his his work on authentication protocols and his contributions to IBM's network security and NetSP, itself an award winning product, in 1994. (The same authentication protocols influenced the later development of the Internet payment protocol, so IBM had to grant a no-fee license, so that the Internet can adopt this payment protocol.) He has been awarded several additional awards and research grants. He has been a member of the Association of Computing Machinery (ACM) since 1992.