Chapter 2

Application Layer Multicast Protocols' Survey

In this chapter, we discuss various existing Application Layer Multicast protocols. Many protocols have been proposed in the recent past, with ESM - End System Multicast [1-2] being the first. Every Application Layer Multicast protocol constructs an overlay on basic network, manages this overlay and transmits information on this overlay. The various protocols that have been proposed differ in three basic criteria - a) the way they create the overlay i.e., topology building algorithm; b) the exact overlay topology they create i.e., mesh, tree, hierarchy; and c) their knowledge about the underlying network topology i.e., fully aware or partially aware or not at all aware of the network. This chapter surveys Application Layer Multicast protocols based on these three criteria and groups them into different categories. Section 1 introduces the Application Layer Multicast with its definition and properties. Section 2 deals with how we classified Application Layer Multicast protocols into various groups and section 3 details specific Application Layer Multicast protocols from each category.

2.1 Introduction

Multicast applications are implemented as multiple point-to-point applications i.e. an application logically requiring multicast must send individually addressed packets to each recipient. This has two drawbacks – 1. The source should know the addresses of all recipients and 2. Transmitting multiple copies of the same packet results in inefficient usage of sender's resources and network bandwidth. Unfortunately, TCP/IP shields the underlying network topology that facilitates multi-destination delivery. For example, local area bus, ring, radio networks and even satellite based wide area networks provide multicast facility directly.

Unicast is completely impractical due to its redundant use of link bandwidth, when thousands of receivers have to receive the same data. The benefits of multicast in terms of bandwidth efficiency are quiet often outweighed by the control complexity associated with group setup and maintenance. Application Layer Multicast protocols normally follow two steps to achieve multicast capability. 1. Arrange receivers into an overlay network of unicast connections and 2. Construct efficient data distribution trees.
over this overlay network. Application Layer Multicast differs from IP Multicast in that the multicast support is done at host systems instead of core routers. Host systems implement all multicast functionalities like membership management, packet replication and data distribution. Multicast in host group model has been defined in [8] as "a host group is a set of network entities sharing a common identifying multicast address, all receiving any data packet addressed to this multicast address by senders that may or may not be members of the same group and have no knowledge of the group's membership" This definition characterizes the following points for the term multicast.

- There must be a unique address to represent the multicast group members.
- All members receive all packets.
- Any member can join the group openly, even without the knowledge of the source.
- Any host can send data to the group irrespective of its membership in the group i.e., non-members also can send data.

Application Layer Multicast differs from traditional multicast in many respects as given below.

- Hosts take over the role of routers for multicasting.
- Within an overlay topology, the underlying physical topology is completely hidden.
- In traditional multicast, the membership knowledge is distributed in the multicast routers. In Application Layer Multicast, group members are known either by a rendezvous point (RP), the source, or everybody or distributed among members.
- The application layer topology is potentially under control.
- The group membership can be under control.
- The precise topology relationship among hosts can control who receives data from whom, and thereby eliminating DOS attacks, which cannot be controlled in traditional IP Multicast by its own definition.

Figure 2.1 shows the distinction between Unicast, IP Multicast and Application Layer Multicast.
Figure 2.1 clearly shows that in case of IP Multicast, packets from source (S) to all hosts (A1, A2, A3), traveled only once on any link. In case of unicast links and routers nearer to source experienced redundant packet movement. In case of Application Layer Multicast, though it does not achieve the performance like IP Multicast, it shows it is better than unicast.

The performance of an Application Layer Multicast can be based on two parameters - stress and stretch [15-17]. Stress is defined per host or per link as 'the number of duplicate packets that a host sends or duplicate packets that travel over a link in an overlay'. Stretch, also called as ‘relative delay penalty’ is defined per pair of hosts i.e., source and destination as the 'ratio between the number of hops a packet travels from source to destination in unicast mode to the number of hops the packet travels in multicast'. In case of IP Multicast, all links experience unit stress and every pair of hosts experiences unit stretch. In case of unicast, in figure 2.1, the maximum stress is 4 on links S-R1 and R1-R2. Stretch in unicast is always 1. In case of Application Layer Multicast, from the figure 2.1, maximum stress is 2 on links R2-A1 and R3-A2; and maximum stretch is 9/4 i.e., 2.5 for the pair S and A3. While unicast shows maximum redundant packet movements on links nearer to source, Application Layer Multicast shows redundant packet movements on links that are nearer to destinations.

Classification of Proposed Application Layer Multicast
The existing Application Layer Multicast protocols can be classified based on three criteria. 1. Knowledge of underlying network topology. This criterion is about the protocol’s awareness of topology to align the physical network topology with overlay topology. 2. Topology building algorithm e.g., distributed or centralized. In a centralized
algorithm, a master host takes decisions about the topology construction, by keeping the physical topology information with itself. In distributed algorithm, every joining host will take its own decision of joining other hosts by its own knowledge about the physical topology. 3. Topology created by the protocol e.g., shared tree, mesh and tree etc.

2.2 Topology Awareness

**Application Layer Multicast** is performed at end hosts, automatically by application code. Performing multicast at end hosts incurs performance penalty, as end hosts do not have the routing information available unlike routers. Unless the overlay topology matches the underlying network topology, the performance would be so degraded that unicast will out perform the gains of multicast. To map overlay topology to the underlying network topology, many application layer multicast protocols rely upon end-to-end measurements to infer network metrics upon which the overlay multicast tree is built. Many a times, the network metrics are based upon rtt (round trip time), traceroute (to get the path), ping (TTL scoped packets) etc. Though, these methods can give some information about network topology, the application layer protocol cannot get the holistic view of the entire network topology.

Based on topology awareness criteria, the proposed application layer multicast protocols are classified as a) **Fully aware** b) **Partially aware** and c) **Not aware**. An application layer multicast protocol is ‘fully aware’ of the underlying network, if it is implemented on network controlled by a single authority, e.g., corporate networks, content distribution networks etc. Over Internet, projects like Topology Server [9] are being implemented that collect entire network information which can be used by an Application layer multicast to align its overlay with the underlying network. An Application layer multicast protocol is ‘partially aware’ of the underlying network if it just guesses the physical connectivity of the routers based on methods like 'rtt' – round trip time; 'ping' - packet internet groper; 'traceroute' - tracing the path between hosts; 'TTL' - time to live scoped messages etc. For example, NICE [15-17] sends TTL scoped messages among the multicast group members to infer the multicast distances among them and uses this information to create 'parent - child' relationships among the members. An Application layer multicast protocol is ‘not aware’ of underlying network topology, if it does not consider any network property like hops, delay, bandwidth etc.,
but rather considers information like public key to create an unique member identity like IP address, for each multicast member. This unique member identity is used to create the overlay topology and distribute the information on this overlay. It may happen that two members who are actually connected by the same router, get such different identities that the information exchange between the two will actually take more time on the overlay topology compared to the time taken when transmitted on physical network topology.

TAG[22] and Appcast (explained in detail in chapter 3) are fully aware of network topology as they depend upon Topology Server[9] that uses network management kind of solutions like polling the routers. HMTP [13], NICE [15-17], ESM [1-2], Yoid [20-21] are partially aware of the topology by depending on methods like trace route, ping etc. These methods construct topology information on their own unlike Appcast and TAG [22] which depend on Topology Server [9]. CAN [18-19], DT Protocol[11], Scribe[14] and Bayeux [12] are not aware of topology at all, as they basically construct overlay topology based on unique member identities that are created randomly. For example, Scribe and Bayeux give separate identity numbers to hosts, based on the hosts’ unique parameters like IP address, Public Key etc. CAN [18-19] and DT Protocol [11] map the hosts into a geometric space by assigning unique random numbers to each member.

Figure 2.4: Classification Based on Topology Awareness
2.3 Topology Building Algorithm

At the heart of Application Layer Multicast protocols, is the overlay topology they create. The topology is created as members/hosts join the multicast group. Application Layer Multicast topology building algorithms define a definite relationship among the participating members and thereby create topologies like tree, mesh, hierarchy etc. The relationship can be parent-child, host-neighbors, cluster member - cluster leader etc. The topology building algorithms differ in the way they build the topology and can be classified as 1. Centralised and 2. Distributed.

The overlay topology is created as the members join the multicast overlay. There has to be a mechanism also called as ‘rendezvous mechanism’ using which the joining member can contact the members of the overlay and become part of the overlay. This ‘rendezvous mechanism’ can be in three ways. 1. Broadcast Mechanism: joining member broadcasts a message containing its willingness to join the overlay and interested overlay members respond to the message. Even for overlay members, when their relationship with immediate parent breaks due to member leave or link failure, to select a new parent, they follow broadcast mechanism. 2. Buddy List: joining members keep a list of 'likely' members of the overlay and contact members from this list. In case of link failures or member leave, the affected overlay members use this list to join the overlay. 3. Well known server: joining members contact well known server to learn about overlay members and join the overlay. Application Layer Multicast protocols that follow the 'broadcast mechanism' or 'buddy list mechanism also termed as limited broadcast' can be termed as 'distributed application layer multicast' protocols and those who follow the 'well known server mechanism' can be termed as 'centralised application layer multicast' protocols. While distributed protocols utilize more bandwidth due to their broadcast mechanisms, centralized protocols suffer from the single point failure of the central server.

2.3.1 Centralised Algorithm

In centralized algorithms, a central server decides the place of a joining host in the topology. A new joining host should contact the central server to know about its own position and its relationship like parent-children etc., with its immediate neighbors. Appcast (explained in chapter 3), DT Protocol [11] and Bayeux [12] fall into this
category. In DT Protocol, new hosts join the overlay by sending a request to 'DT Server'. The server responds with the logical and physical addresses of some host that is already in the overlay network. The new host then sends a message to the host identified by the server and thus establishes its relationship with the overlay. In Bayeux, the joining host sends a 'JOIN' message to 'Root' i.e., central server. 'Root' replies with a 'TREE' message. While the 'TREE' message is traveled towards the joining host, all in between hosts include the joining host's ID in their list of hosts for whom they have to route the information, creating a routing path from the root to the joining host.

2.3.2 Distributed Algorithm

In distributed algorithms, all joined hosts guide the new joining host, to know its place in the topology and relationships. The onus of contacting the already joined members and knowing its own position is on the joining host only. NICE[15-17], Yoid[20-21], HMTP[13] etc. fall into this category of protocol as shown in figure 2.3. In Narada, every host contacts a common host (called RP - Rendezvous Point) before joining the group. This common host returns addresses of a set of already joined hosts. New host contacts them randomly to find whether any one will accept it to become its neighbor. The one who accepts it first will be given first priority. In NICE, 'root' periodically sends a heartbeat packet to all overlay members, from which each member infers their distance to 'root'. Similarly, each parent sends messages to its children from which children infer their distance to parent. Using this information, members reorganize themselves in the overlay.

Figure 2.5: Classification Based on Topology Building Algorithm
2.4 Topology Created

Application layer multicast protocols create and manage the overlay topologies to efficiently route the data packets. Based on the topology they create, the proposed protocols can be classified as those who create a. Tree, b. Mesh or c. Clusters. All Application Layer Multicast protocols that create 'Tree' topology ensure that there be no loop in the path between any pair of participating nodes. In contrast, the protocols that create 'Mesh' topology look for alternate paths between nodes. In case of 'cluster' topology, the members are grouped in clusters and one of these members acts as a cluster leader. The cluster leaders are grouped again into clusters at higher level and one of them acts as the cluster leader at the level. The cluster grouping into levels continues, till only one member becomes the leader of the entire cluster hierarchy. The general purpose of creating a topology is 1. To distribute the data packets and 2. To send control information to manage the topology. Protocols like HMTP [13], TAG [22], Appcast use the same topology for both the purposes, while others like Narada [1-2], Yoid [20-21] use separate topologies like tree and mesh.

Figure 2. 6: Classification Based on Topology Created

2.4.1 Mesh topology
End System Multicast (ESM)/Narada [1-2], Scattercast [3-4] and Yoid [20-21] fall into this category. Data topology is tree in all these systems. ESM constructs mesh first as the
basic overlay network and then constructs separate trees for each source. Narada(ESM) and Scattercast construct trees in a two-step process. First they construct efficient meshes among participating members and then construct spanning trees of the mesh using well-known routing algorithms. Yoid constructs tree first and then constructs mesh to support this tree. Unlike ESM, Yoid uses the single shared tree for all sources for data dissemination. DT protocol [11] uses mesh for both data and control topologies.

2.4.2 Tree
Protocols like Overcast [5], TAG–Topology aware group communication [22], ALMI–Application Level Multicast Infrastructure [10] and HMTP-Host Based Multicast [13] create shared tree. Single shared trees are not optimized for the individual source and are prone to failures as a single point of failure can break the tree into major partitions. In case of mesh, a link failure can with stand due to alternate paths. Only multiple failures can cause the mesh to partition. The single shared tree approach, justifies its method from the fact that the underlying network topology already takes care of multiple paths. So one should consider node failures than the link failures. An efficient way is to maintain multiple trees for each source, but that requires multiple overlay topologies, which is complex. The application layer multicast protocols that create trees as their basic topologies differ in the number of trees they manage per source, per group or a single shared tree.

2.4.3 Clusters
Protocols like CAN and NICE group the members into clusters. Each cluster is assigned a cluster leader. The cluster leaders in turn are grouped and one of the cluster leaders is assigned as leader of this group. This process repeated till there exists only one leader on top of the grouped cluster leaders, generating a hierarchy topology. NICE arranges set of end hosts into a hierarchy. The hierarchy implicitly defines data path. Each member maintains soft state information about other hierarchically nearer members and has only limited knowledge about other members. In NICE, all members belong to Layer 0. Members are grouped into clusters with size K and 3K where K>1. For each cluster, one of the cluster members acts as a leader and enters into higher layers.
2.5 Well-known Application Layer Multicast Protocols

In this section, we describe certain well-known protocols in detail, to support our classification discussed in previous section.

2.5.1 ESM - End System Multicast / Narada

ESM [1-2] is one of the first application layer multicast protocol. Narada constructs an overlay structure among participating end systems in a self-organizing and fully distributed manner. Narada creates an overlay mesh topology on top of router topology connecting all hosts that participate in a multicast session. Once the mesh is created, any one from the group can send messages to every one else in the group. For this, Narada uses existing multicast protocols like DVMRP – Distance Vector Multicast Routing Protocol. Narada has protocols for host join, mesh partition repairing, adding/deleting mesh links for better performance.

**Member Join:** When a member wishes to join a group, Narada assumes that the member is able to get a list of group members by an out-of-band bootstrap mechanism. The list needs neither be complete nor accurate, but must contain at least one currently active member. New host contacts members in list, randomly to find whether any one will accept it to become its neighbor. A host is a neighbor to another host, if they have a direct connection on the overlay. It repeats this process till some host accepts it as a neighbor. The one who accepts it first will be given first priority. Having joined the group, the new member exchanges refresh messages with neighbors who in turn exchange with their neighbors. This process keeps on going till every one in the group is aware of the new member join.

**Member Leave:** When a member leaves a group, it notifies its neighbors and this information is propagated to the rest of the group members along the mesh.

**Control Messages:** The group membership information keeps flowing across all members of the group periodically. Every host keeps state information about every other host.

The complexity of Narada algorithm is $O(N^2)$, where N is the number of hosts. Narada cannot scale for larger groups. As the group's size increases, the number of overlay hops between any pair of members' increases and hence the delay between them potentially increases.
2.5.2 Yoid - Yet Other Internet Distribution Protocol

Yoid [20-21] generates two topologies per group - a shared tree topology and a mesh topology. The tree topology is for data distribution and the mesh topology is to support the tree from not breaking. While Narada creates mesh first and then tree; yoid creates tree first and then mesh. Yoid has many protocols on top of TCP and UDP like YIDP - Yoid Identification Protocol, The Yoid Transport Layer - yTCP, yRTP, yMTCP etc., and Yoid Distribution Protocol - YDP.

**Yoid Tree:** In Yoid, every host is responsible to find its parent. A new host gets a list of potential parents from a common host (RP). A new host selects one as its parent if the parent-child link does not cause a loop. If a new host finds many parents, it will select best based on some metric of interest like number of hops from parent to child or number of children the parent already has etc. If new host could not get any parent, it declares itself as parent and informs RP. RP arbitrates and merges the tree partitions into a single tree.

**Yoid Mesh:** After constructing the tree, Yoid proposes to have multiple paths among members such that they will act as back up. So each host randomly selects few more members in the group, which are not its immediate neighbors and constructs links in a different path. These additional paths will give strength to Yoid in case of failures. Yoid has many disadvantages like Yoid is too complex, the specifications are too descriptive and lengthy, too many protocols and APIs and all these issues will actually burden the whole set up.

2.5.3 HMTP - Host Multicast

HMTP [13] creates group specific tree topology as the multicast overlay topology. In HTMP, each multicast group requires a Host Multicast Rendezvous point that acts as a contact point for new members to join the group. HMTP Clusters nearby members together. Members choose their parent closer to them. Following steps describe the HMTP protocol.

1. New member sets the root as potential parent (PP) and contacts PP.
2. Query PP to discover all its children and measure its nearness to PP and PP's children.
3. Find the nearest member among the PP and PP’s children except those marked as invalid. If all of them are marked as invalid, pop the top element from stack, set it as PP and return to step 2.

4. If the nearest member is not current PP, push current PP onto stack; set the nearest member as new PP and return to step 3.

5. Otherwise send PP join request. If PP accepts it as a child, it becomes child of the PP; if rejected mark PP as invalid and return to step 3 (PP may not accept it as its child due to many reasons like - out degree); otherwise parent found and so establish unicast path.

HMTP proposed member leave, link failure and improvement algorithms also. In HMTP [13], every member keeps track of every other member that falls in the path of member and root.

2.5.4 NICE - Nice Internet Communication Environment

NICE [15-17] claims, relatively small control overhead. Its motivation is actually from key distribution in a secure group communication. NICE arranges set of end hosts into a hierarchy. The hierarchy implicitly defines data path. Each member maintains soft state information about other hierarchically nearer members and has only limited knowledge about other members. In NICE, all members belong to Layer 0. Members are grouped into clusters with size K and 3K where K>1. For each cluster, one of the cluster members acts as a leader and enters into higher layers. A member is part of L_i if it is leader in all L_i-1 levels. A cluster leader has minimum maximum distance from all of its members. A host belongs to only a single cluster at any layer. If a host is not present in layer L_i, it cannot be present in any layer L_j, where j>i. For a group size N, and cluster size K, there can be at most log_K N layers. Each member maintains information about every other member of its own cluster in all of its layers.

NICE [15-17] constructs an overlay tree, before it clusters the group members and arranges them into a hierarchy. NICE constructs an overlay tree based on the underlying network topology. Next, it uses a clustering protocol to group the members into clusters of size K to 3K-1, where K is a constant by traversing the overlay tree bottom up. This clustering is basically to reduce the depth of the tree and to keep control overhead cost to
be constant. As the cluster size increases, unicast with in the cluster may increase. NICE [15-17] does not give flexibility to the joining member, to choose its leader.

**Member Join:** When a new host joins the multicast group, it must be mapped to some cluster in layer First, it contacts the RP with its join query. The RP responds with the hosts that are present in the highest layer of the hierarchy. The joining host then contacts all members in the highest layer to identify the member closest to it. Once a nearest host (CL\_i) is found in a layer, the joining host contacts all child members of CL\_i in (i-1) layer, to identify the member nearest to it. This process is repeated till it gets a cluster to join as a member.

**Member Leave:** When a host leaves the multicast group, it sends a Remove message to all clusters to which it is joined. This is a graceful-leave. However, if host H fails without being able to send out this message, all cluster peers of H detects this departure through non-receipt of the periodic HeartBeat message from H.

**Cluster Split and Merge:** A cluster-leader periodically checks the size of its cluster, and appropriately splits or merges the cluster when it detects a size bound violation. A cluster that just exceeds the cluster size upper bound, 3k-1 is a split into two equal-sized cluster. Subsequently, an equal-sized split would create two clusters of size k each. However, a single departure from any of these new clusters would violate the size lower bound and require a cluster merge operation to be performed.

### 2.5.5 CAN - Content Addressable Network

CAN[18-19] is designed for large groups. While other methods are towards data dissemination initiated from a single source, this method also tries to achieve interactive group games. CAN[18-19] maintains a logical d-dimensional co-ordinate space, with no relation to physical co-ordinate system or network topology. The entire co-ordinate space is dynamically partitioned among all the nodes. Every node, owns its individual unique zone with in the overall space. The new node must find a node already in the CAN[18-19]. Using the CAN[18-19] routing mechanism, it must find a node, whose zone will be split. The neighbors of the split zone must be notified so that routing can include the new node.

**Member Join:** To find a zone, new node randomly chooses a point (x,y). It sends join request destined for point (x,y). This message is sent into the CAN via any existing
CAN node. Each CAN node then uses the CAN routing mechanism to forward the message, until it reaches the node in whose zone \((x, y)\) lies. This current occupant node then splits its zone in half and assigns one half to the new node. Having obtained its zone, the new node learns the IP addresses of its coordinate neighbor set from the previous occupant. This set is a subset of the previous occupant's neighbors, plus that occupant itself. Similarly, the previous occupant updates its neighbor set to eliminate those nodes that are no longer neighbors. Finally, both the new and old nodes' neighbors must be informed of this reallocation of space. Every node in the system sends an immediate update message, followed by periodic refreshes, with its currently assigned zone to all its neighbors. These soft-state style updates ensure that all of their neighbors will quickly learn about the change and will update their own neighbor sets accordingly.

Member Leave: When nodes leave a CAN, the zones they occupied are taken over by the remaining nodes. The normal procedure for this zone take over is for a node to explicitly hand over its zone and the associated (key, value) database to one of its neighbors. If the zone of one of the neighbors can be merged with the departing node's zone to produce a valid single zone, then this is done. If not, then the zone is handed to the neighbor whose current zone is smallest, and that node will then temporarily handle both zones.

2.5.6 Bayeux

Bayeux [12] is an efficient, source specific, explicit join application level multicast system. It uses Tapestry [12], an application level routing protocol. Each Tapestry node has names independent of their location and semantic properties, in the form of random fixed length bit sequences represented by a common base (e.g., 40 hex digits represented by 160 bits). Bayeux uses four types of control messages in building a distribution tree - JOIN, LEAVE, TREE and PRUNE. A member joins the multicast session by sending a JOIN message towards the 'Root'. 'Root' replies with a TREE message. The actual paths taken by JOIN and TREE are different due to asymmetric nature of Tapestry unicast routing. When a router receives TREE message, it adds the new member node ID to the list of receiver node IDs that is responsible for, and updates its forwarding table. LEAVE message from an existing member triggers a PRUNE
message from the root, which trims from the distribution tree any routers, whose forwarding state become empty after LEAVE operation.

**Member Join:** When a node with Id 1250 tries to join multicast session where node 7876 is the root, a JOIN message from node 1250 traverses nodes xxx6, xx76, x876, and 7876 via Tapestry unicast routing, where xxx6 denotes some node that ends with 6. The root 7876 then sends a TREE message towards the new member traversing the nodes xxxO, xx50, x250 and 1250, which sets up the forwarding state at intermediate application-level routers. Note that while both control messages are delivered by unicasting over the Tapestry overlay network, the JOIN and TREE paths might be different, due to the asymmetric nature of Tapestry unicast routing. When a node receives a TREE message, it adds the new member node ID to the list of receiver node IDs that it is responsible for, and updates its forwarding table. For example, consider node xx50 on the path from the 'root node to node 1250. Upon receiving the TREE message from the root, node xx50 will add 1250 into its receiver ID list, and will duplicate and forward future packets for the multicast session to node x250.

**Member Leave:** A node that leaves the overlay, informs the root by sending a LEAVE message. A LEAVE message from an existing member triggers a PRUNE message from the root, which trims from the distribution tree any routers whose forwarding states become empty after the leave operation.

**2.5.7 TAG - Topology Aware Group Communication**

TAG [22] uses information about path overlap among group members to construct the overlay tree. In TAG each new member of multicast group, determines the path from the root to itself and finds out its parent and children by partially traversing the overlay tree. TAG proposed complete path matching algorithm, where in a new node selects one as its parent, which shares the maximum common path with it. Each TAG node maintains a Family Table, with information about its parent and children. The path-matching algorithm traverses the overlay tree from root down the children, matching the paths from the 'root to new node' with the path from 'root to TAG node'. It considers three mutually exclusive cases. Let N be a new member wishing to join and C be the node being examined. Then the three cases are 1. There exists a child A of C, whose path is a prefix for the path N, with the condition that the path length of N > A > C. In this
case N chooses node A, and traverses the sub-tree rooted at A. 2. There exist children $A_i$ of $C$, who have the path of N as the prefix, in their path. In this case, N becomes child of C, with all A; as its children. 3. In case, there's no child of C satisfying the cases 1 or 2, N becomes the child of C. As an optimization method, TAG [22] proposed partial path matching algorithm, where in, instead of matching the complete path of a new member, a predefined number of elements in the path are matched. This helps reduce the depth of the tree.

**Member Join:** A new member joining a session sends a JOIN message to the main sender S of the multicast session (the root of the tree). Upon the receipt of a JOIN, S computes the 'spath' (shortest path), to the new member, and executes the path-matching algorithm. If the new member becomes a child of S, the FT (family tree) of S is updated accordingly. Otherwise, S propagates a FIND message to its child that shares the longest 'spath' prefix with the new member 'spath'. The FIND message carries the IP address and the 'spath' of the new member, and is processed by executing path matching and either updating the FT, or propagating the FIND. The propagation of FIND messages continues until the new member finds a parent.

**Member Leave:** A member can leave the session by sending a LEAVE message to its parent. A LEAVE message includes the FT of the leaving member. Upon receiving LEAVE from a child, its parent removes the child from its FT and adds FT entries for the children of leaving child.

### 2.6 Conclusions

In this chapter we have looked at different Application Layer Multicast protocols. We analyzed these protocols and could classify them based on properties like a) the topology they create – tree, mesh, hierarchy etc., b) kind of algorithm they use like centralized, distributed etc., and c) based on their awareness of network topology. While we could classify the protocols and explain them in this chapter, we simulated these protocols and compared their performances and time complexities of the algorithms along with our proposed protocol ‘Appcast’ in chapter 3.