End System Multicast: An Architectural Infrastructure and Topological Optimization ⋆

Starsky H.Y. Wong and John C.S. Lui [1]

Abstract

Although IP-multicast has been proposed and investigated for years, there are major problems inherent in the IP-multicasting technique, e.g., difficulty to scale up the system, difficulty in allocating a globally unique multicast address, complexity in supporting higher level features such as reliable data transfer and congestion/flow control, more importantly, difficulty to deploy on the current Internet infrastructure due to necessity to change many core routers. Recently, End-System Multicast (ESM) has been proposed as an alternative solution so that multicasting services can be quickly deployed. In this paper, we consider the "architectural" and "optimization" issues on designing an ESM-tree. Specifically, we present a distributed algorithm on how to create and maintain an ESM-tree. We propose a distributed algorithm to perform tree optimization (TO) so that an ESM-tree can dynamically adapt to the changing network condition (e.g., drop in transfer bandwidth) so that the nodes within an ESM-tree can receive data more efficiently. The distributed algorithm has the important theoretical properties that at all times, a tree-topology can be maintained and any node joining, leaving, as well as any tree optimization operation will not "partition" the underlying ESM-tree. Therefore, our work can be used to provide an efficient architectural infrastructure for ESM services. We have implemented a prototype ESM system and carried out experiments to illustrate the effectiveness and the performance gains of our ESM optimization protocol.

Key words: end system multicast, topology optimization, distributed algorithms

1 Introduction

Multicasting is a mode of communication between a sender and many receivers. The main advantage of multicasting is that a sender only needs to send the data once so that significant network transmission resources can be saved. IP multicasting[9,12,13,15,26,32,28] is a conventional way to provide the multicasting services over IP networks. To support IP multicasting, routers within the IP networks need to be "modified" so as to maintain many multicast state informations, e.g., membership for each multicast group, input/output ports for each multicast group so as

⋆ This work is supported in part by the RGC Research Grant.

[1] Dept. of Computer Science & Eng, The Chinese University of Hong Kong, Shatin, Hong Kong. Email: cslui@cse.cuhk.edu.hk. Corresponding author: John C.S. Lui

to perform proper packet forwarding, packet error recovery and congestion control within a multicast group.

There are major problems[10,14,40] in deploying the IP multicast on the Internet. IP multicast requires the core routers to maintain multicast group membership. This not only violates the *"stateless"* principle of the original Internet design, but also introduces high design/implementation complexity on routers. A *"stateful"* IP multicast router [17] implies a major scalability problem[17]. Also, IP multicast requires each multicast group to obtain a globally unique IP multicast address for communication but this unique address allocation is difficult to ensure in a distributed, scalable and consistent manner. Also, to multicast data in a reliable and secure fashion, router needs to participate in the error recovery [20,42] and congestion control processes[4,5,24,**?**,23,34,35,37,39]. Since not all routers in the Internet are IP-multicast enabled, this creates a major deployment problem.

One way to overcome the problems described above and to deploy the multicasting service quickly is to use the *end-system multicast* (ESM) approach[6,10,19,21,44,33]. In essence, an ESM is an approach to rely on end hosts to provide all multicast related functions, such as group management and multicast routing based on IP unicast. The main advantage of ESM over the IP multicast is that ESM does not require core routers support and hence resolve the deployment problem. To realize an ESM service, most multicasting functionalities are pushed up to the end systems, instead of relying the support from the core routers.

Although authors [1,2,10,11,30,41,7] demonstrate the flexibility and advantages of using an ESM to deliver multicasting services, there are still many unresolved issues. For example:

(1) What is the proper software architecture to manage the group membership?
(2) How to make sure that an ESM topology is a tree structure [2] *so as to have efficient group communication?*
(3) How end system can adapt to the changes of network condition (e.g., sudden drop in network bandwidth) and still be able to deliver information efficiently to all members?

All these issues require a careful architectural and software design so as to avoid problems such as distributed deadlock and data inconsistency. The contribution of our work is that we consider "architectural" and "optimization" issues on designing an ESM-tree. Specifically, we present a distributed algorithm on how to create and maintain an ESM-tree. We propose a distributed algorithm to perform tree optimization (TO) so that an ESM-tree can dynamically adapt to the changing network

[2] A non-tree structure implies that some data will be sent in a redundant fashion and thereby consuming more network resources. On the other hand, a non-tree structure can provide redundant paths so as to enhance reliability. Note that our ESM system can be extended to mesh structure by simply taking the union of multiple trees.

condition, e.g., drop in transfer bandwidth, so that nodes within an ESM-tree can receive the data more efficiently. The proposed distributed algorithm has the important theoretical properties that at all times, a tree-topology can be maintained and any node joining, leaving, as well as any tree optimization operation will not "partition" an ESM-tree. Therefore, our work can be used to provide an efficient architectural infrastructure for ESM services.

The outline of the paper is as follow. In Section 2, we present our architectural as well as different components of an ESM system. In Section 3, we present the distributed algorithm for the ESM-tree formation, data transfer, tree optimization and node leaving protocol. In Section 4, we carry out experiments on our prototype system as well as NS2[31] simulation to illustrate the functionalities as well as the performance of the proposed ESM architecture. Related work is presented in Section 5 and conclusion is given in Section 6.

2 System Architecture

In our proposed ESM system, an end system (or end host) is represented as a node in an ESM-tree. There are three different types of nodes in an ESM-tree, they are: i) a *root node* (N_R), ii) a *bootstrap node* (N_b), and iii) any participating *client node* (N_i for $i = 1, 2, \ldots$). The root node N_R is the source of data and it is responsible for initiating the multicast session. For the ease of presentation, we assume that there is only one root node in an ESM system. Note that the proposed algorithm can easily accommodate an ESM-tree with multiple source nodes. The root node has a fan-out constraint ($\mathcal{F}_{N_R} \geq 1$), which limits the number of *directly* connected client nodes. To initiate an ESM session, a root node needs to register with a specific bootstrap node. A bootstrap node N_b is a well-known server that stores the group information about a multicast session. For example, it stores the root node's ID (e.g., IP address) as well as ID of any client node in an ESM-tree. Whenever a new client (let say N_i) wants to join an ESM session, it first contacts the N_b node. Under the ESM architecture, a client node may also play a role of a sender to other client nodes. For each client node, there is a fan-out constraint, which is denoted by $\mathcal{F}_i \geq 1$. Again, this sets the upper bound on the number of client nodes that can be attached to the node N_i.

Since the network conditions such as available bandwidth and transmission delay are changing from time to time, to ensure the efficient operation of an ESM system, each client node will periodically test whether the current data transfer bandwidth from its parent node is satisfactory or not. If the transfer bandwidth is not satisfactory, then a client node will initiate a *tree optimization(TO)* operation so as to find another parent node that can provide a higher transfer bandwidth. We will address this operation in detail in Section 3.

The high level operation of our ESM system is as follow. The root node N_R first contacts a well-known bootstrap server N_b for the ESM initialization. A client node

N_i can participate in the multicast service by first joining the ESM-tree. This is accomplished by first contacting the bootstrap node N_b. In return, N_b replies a list of potential clients to N_i when N_i wants to attach to an ESM-tree. The client node N_i then chooses a parent node from this returned list. After the successful attachment to the ESM-tree, the client node N_i can receive data from its parent. Data transfer is accomplished in a "pipeline" fashion, that is, a client node plays a role as a sender *and* a receiver at the same time (except those clients nodes which are the leaf nodes in an ESM-tree). Also, a client node N_i may choose to find a new parent node if the transfer bandwidth from its parent node is below some predefined threshold. In this case, tree optimization operation will be invoked. The *main challenge* of designing an ESM system is to make this distributed system *"scalable"* and *"consistent"*, e.g., without deadlock and loop formation. Again, we will explain in detail the operations and protocols of the propose ESM system in Section 3.

We made the following assumptions about our proposed ESM system: 1) nodes in the ESM-tree can communicate with each other by exchanging control messages only (e.g., via TCP). 2) Control messages will not be lost or altered and are correctly delivered to their destination nodes in a finite amount of time. 3) Control messages will be delivered in the order they are sent. 4) Each node has a first-in-first-out queue to store the arrived control messages and they will be processed in a first-come-first-serve manner, and 5) data transfer between nodes can be carried out using either the TCP or UDP protocols.

3 ESM Protocols

In this section, we describe various ESM protocols [3]. In Table 1, we first define various notations which will be useful for the discussion on the ESM system consistency via the distributed locking operations. At any time, the ESM management protocol ensures that any node N_i can only be in one of the following states: (1) Both $L_{i,LR}$ and $L_{i,LP}$ are empty, or (2) $L_{i,LR}$ is not empty and $L_{i,LP}$ is empty, or (3) $L_{i,LP}$ is not empty and $L_{i,LR}$ is empty. This property is to ensure that there can be no "loop" within an ESM-tree and thereby eliminate the possibility of an ESM-tree partition event.

3.1 ESM: The Tree Formation Protocol

When a node N_i wants to join an ESM-tree, N_i first gets a partial ESM-tree topology from the bootstrap server. Then, N_i finds a *potential* parent node, say node N_j, from this partial ESM-tree topology. After that, N_i tries to take a "LP Lock" in the potential parent node. In essence, a $LP_{j,i}$ is a lock to indicate that node N_i wants to attach to node N_j. Finally, N_i makes a real connection to its new parent, node N_j.

[3] For examples and illustrations of these protocols, please refer to [38]

Notations	
N_b	the well-known bootstrap server
N_R	the root node
N_i	a client node with identifier i (i may be IP address, host name ... etc.)
$\mathcal{F}_i$	the "fan-out" limit of a node N_i.
NC_i	number of children nodes which are connected to the client node N_i.
P_i	the parent node of N_i.
T_i	the sub-tree rooted at node N_i.
$List_i$	$List_i$ is a list in node N_i which contains information of other nodes in an ESM-tree. Each entry in $List_i$ has the form of $[N_j$, IP address of $N_j, \mathcal{F}_i]$.
$LR_{i,j}$	$LR_{i,j}$ is a lock indicates that node N_i is locked by node N_j, where N_j is an ancestor of N_i. Therefore, N_i cannot be a new parent node for other nodes in an ESM-tree. We use this lock for the tree optimization operation.
$LP_{i,j}$	$LP_{i,j}$ is a lock indicates that node N_i is currently locked by node N_j. Therefore, N_i is a potential parent node for N_j.
$LW_{i,j}$	$LW_{i,j}$ is a lock indicates that node N_i is currently locked by node N_j with "LW" type of lock. After that, N_i needs to reject all the new coming "LP" and "LR" locking request.
$L_{i,LR}$	the set of LR locks that are taken on node N_i.
$L_{i,LP}$	the set of LP locks that are taken on node N_i.
$L_{i,LW}$	the set of LW locks that are taken on node N_i.

Table 1
Notation for ESM

The procedure for a client node N_i to join an ESM-tree is described as follow:

Procedure join_ESM(INPUT:address of bootstrap server, OUTPUT:NULL)
01 {
02 **while** N_i **is not connected to the ESM-tree** {
03 /* use some selection criteria for selecting */
04 /* a sub-list from bootstrap nodeN_b */
05 **contact** N_b **to get sub-list of** $List_b$**;**
06 $tempList_i <-$ **sub-list of** $List_b$**;**
07 $n <- |tempList_i|$**;** /* number of potential parent node */
08 **sort** $tempList_i$ **according to performance metric (e.g delay or available BW) ;**
09 **for** k **from 0 to** $n-1$ {
10 **send "join" request to** $tempList_i[k]$**;** /* sends LP locks */
11 **wait for reply from** $tempList_i[k]$**;**
12 **if(reply == success)**{
13 /* $tempList_i[k]$ is the new parent node */
14 P_i = $tempList_i[k]$**;**
15 /* receive the ESM-tree topology from parent */
16 **receive** $List_{P_i}$ **from** P_i**;**
17 $List_i <- List_{P_i}$**;**
18 N_i **is connected to the ESM;**
19 **break;**

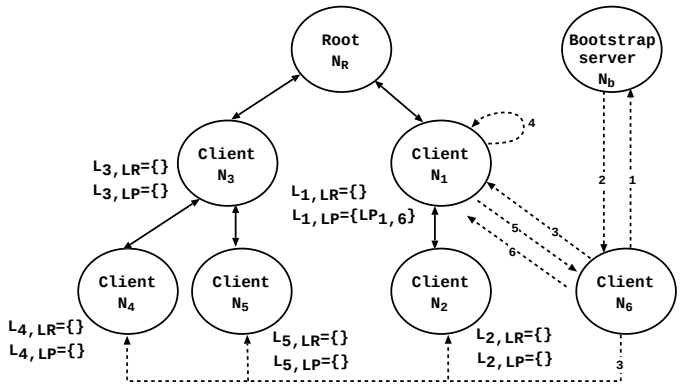

Fig. 1. Bootstrap procedure

```
20      }
21    }
22  }
23  /* broadcast Ni is part of the tree */
24  Ni update status via flooding;
25 }
```

In the above procedure, $tempList_i \subset List_b$ contains a subset of nodes IDs stored in N_b. Various methods can be used to select $tempList_i$ from $List_b$, for example: (1) randomly select a node from $List_b$, (2) select those nodes in $List_b$ that have not reached the fan-out limit, or (3) use "IP address" and subnet mask to select nodes in $List_b$ such that they are within the geographical neighborhood of N_i. Node N_i can sort all nodes in $tempList_i$ according to some performance metrics. For example, one can use the packet train techniques[3,16,25] to determine the available bandwidth between N_i and its potential parent nodes or node N_i can use the "ping" utility to estimate the round trip delay between its potential parent node.

After knowing the connectivity condition of these nodes, N_i will sort them and contact one of the node with the highest performance measures. If the node N_j, which N_i contacted, can admit N_i as its children, N_i will receive an ESM-tree topology information from N_j (N_i's new parent) and become one of the child of N_j. Finally, N_i will send an ESM broadcast message (via flooding) to inform other nodes within the ESM-tree that it had became a child of N_j.

Figure 1 illustrates a scenario wherein the ESM-tree has a root node N_R and five client nodes (N_1 to N_5). A new client node N_6 wants to join the multicast session. N_6 first sends a request to N_b to get the current information of the ESM-tree. The bootstrap node N_b returns a *subset* of client nodes which are currently connected to the ESM-tree. Given these subset of nodes, N_6 can determine which node is the most favorable parent node by testing the available transfer bandwidth or delay be-

tween these potential parent nodes and N_6. Assume that N_6 wants to join N_1, it then sends a *"join request"* to N_1. Upon receiving the "join request", a node, say N_j, executes the following procedure:

Procedure reply_join_request(INPUT:NULL,OUTPUT:NULL) {
01 {
02 **if** N_j **receive a "join" request from** N_i {
03 /* Test whether some ancestors of N_j locked the node N_j by LR */
04 **if(** $|L_{j,LR}|$ **!=** 0 **)**
05 **send "fail" to** N_i**;**
06 /* N_j or ancestors of N_j want to leave */
07 **else if(** $|L_{j,LW}|$ **!=** 0 **)**
08 **send "fail" to** N_i**;**
09 /* N_j had reach its Fan-out limit */
10 **else if(** NC_j + $|L_{j,LP}| >= \mathcal{F}_j$**)**
11 **send "fail" to** N_i**;**
12 **else** {
13 **Add** $LP_{j,i}$ **into** $L_{j,LP}$
14 **send "success" to** N_i**;**
15 NC_j**++;**
16 /* Tell the new child current ESM-tree Topology */
17 **send** $List_j$ **to** N_i**;**
18 **Remove** $LP_{j,i}$ **from** $L_{j,LP}$
19 }
20 }
21 }

In the example of Figure 1, N_1 checks whether $L_{1,LR}$ is empty or not. If $L_{1,LR}$ is not empty, which implies that some other nodes try to select N_1 as parent, then N_1 should reject the join request. Then, node N_1 checks whether $L_{1,LW}$ is empty or not. If $L_{1,LW}$ is not empty, which implies that some ancestors of N_1 want to leave the ESM-tree, then N_1 should reject the join request also. When node N_6 receives a rejection message from node N_1, node N_6 can choose other node from the subset list as its potential parent and the whole process repeats. If $L_{1,LR}$ is empty, then N_1 checks the following condition of $|L_{1,LP}| + NC_1 < \mathcal{F}_1$. This condition implies that the fan-out limit of N_1 has not been reached. If the condition is not satisfied, then N_1 has to reject the join request. If the condition is satisfied, then N_1 adds $LP_{1,6}$ into the set $L_{1,LP}$ and sends an accept message back to N_6. After this, N_1 sends its $List_1$ back to N_6 because the $List_6$ in N_6 is only a sub-list of $List_{N_b}$. Node N_6, upon receiving the accept message, needs to broadcast the information of its

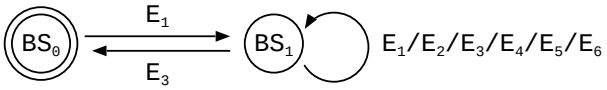

Fig. 2. State Transition Diagram of a Bootstrap Server for Tree Formation

attachment to N_1 to the whole ESM-tree.

Theorem 1 *The above distributed algorithm for join procedure by a client node guarantees that the ESM is a tree topology.*

Proof: We can show that (1) ESM topology is a connected graph, and (2) the topology is always a tree. For the first case, a newly arriving client node always contacts the bootstrap server N_b that replies with a list of potential client nodes from the ESM connected topology. The newly arriving node will eventually select one of these nodes as its parent and the new node will be part of the connected graph. Therefore, a connected graph is maintained after a join operation. To show that the ESM topology is a tree, we can easily show it by contradiction. Consider a client node N_i with multiple parent nodes. This will only occur if the client node N_i sent out multiple attachment requests and received multiple positive replies. However, this case would not occur because the join procedure listed above only attempt to make one attachment at a time. Therefore, node N_i has only one parent and the resulting ESM topology is a connected tree. ∎

Theorem 2 *The above distributed algorithm for join procedure by a client node guarantees that there is no partition in the ESM-tree.*

Proof: Assume the contrary, tree partition will result from join procedure. This implies that the following situation will occur: tree partition occurs when two or more nodes, that are not connected to ESM-tree, join up themselves rather than connect to the ESM-tree. Without loss of generality, assume there are two nodes, N_x and N_y. N_x and N_y are going to join the ESM-tree and eventually they connect to each other. Base on the bootstrap procedure, both nodes will first get a sub-list of node, $tempList_x$ and $tempList_y$, from the bootstrap server. The sub-list of node, $tempList_x$ and $tempList_y$, returned from the bootstrap server must contain the information of either N_x or N_y. That is, N_x or $N_y \in tempList_x \cup tempList_y$ Otherwise, N_x and N_y cannot connect to each other. However, this is impossible because the bootstrap server will not contain information of nodes that have not joined the ESM-tree (i.e. information of N_x and N_y). This contradicts our basic requirement. ∎

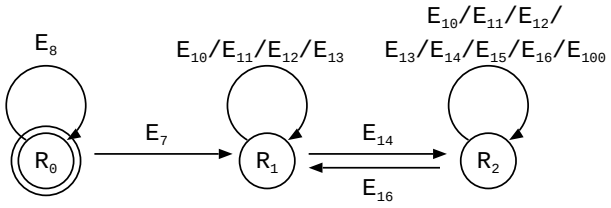

Fig. 3. State Transition Diagram of a Root node for Tree Formation

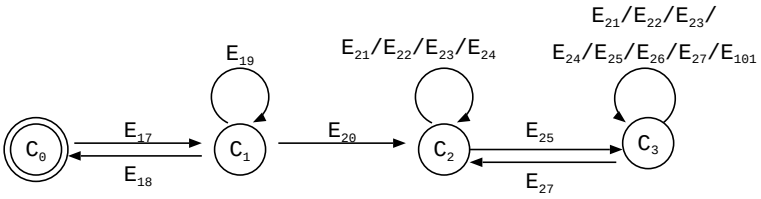

Fig. 4. State Transition Diagram of a Client node for Tree Formation

State	**Description**
BS_0	The initial state of a Bootstrap server. The number of ESM-tree is 0.
BS_1	This is the normal state of a Bootstrap server. The number of ESM-tree is greater than 0.

Event	**Description**	**Receive status of the bootstrap server**	**Messages sent by the bootstrap server**
E_1	The bootstrap server receives a "create ESM-tree request" from a root node and replies a "success" message back to the root node. After this, the number of ESM-tree is increased by 1.	"create ESM-tree request"	"success" message for the "create ESM-tree request"
E_2	The bootstrap server receives a "create ESM-tree request" from a root node and replies a "fail" message back to the root node. This may imply that too many ESM-trees are registered.	"create ESM-tree request"	"fail" message for the "create ESM-tree request"
E_3	The bootstrap server receives a "remove ESM-tree request" from a root node. After this, the number of ESM-tree is decreased by 1.	"remove ESM-tree request"	Nil
E_4	The bootstrap server receives a "attach to ESM-tree request" that is initiated by node N_i and replies a partial ESM-tree topology back to N_i. This implies that N_i wants to join the ESM-tree.	"attach to ESM-tree request"	partial ESM-tree topology
E_5	The bootstrap server receives an "addition of node information request" that is initiated by node N_i, After this, $List_{BS}$ is updated according to the information received.	"addition of node information request"	Nil
E_6	The bootstrap server receives a "removal of node information request" that is initiated by node N_i. After this, $List_{BS}$ is updated according to the information received.	"removal of node information request"	Nil

Table 2
Description of State Transition Diagram in Figure 2 for different states and events.

3.1.1 State Transition Diagram for Tree Formation Protocol

In the following, we use finite state machine representation to formally discuss the actions by various nodes during the ESM-tree formation process. Figure 2 to Fig-

State	Description
R_0	The initial state of a root node. It has not registered in any bootstrap server.
R_1	This is the normal state of a root node. It has registered in a bootstrap server.
R_2	This is a "locked" state of a root node. It is locked by one or more "LP Locks" (i.e. $\|L_{R,LP}\| \geq 1$). This implies that some nodes may want to become a children of the root node.

Event	Description	Receive status of the root	Message sent by the root
E_7	The root node sends a "create ESM-tree request" to the bootstrap server and the bootstrap server replies a "success" message. After this, a new ESM-tree is formed.	"success" message for the "create ESM-tree request"	"create ESM-tree request"
E_8	The root node sends a "create ESM-tree request" to the bootstrap server and the bootstrap server replies a "fail" message.	"fail" message for the "create ESM-tree request"	"create ESM-tree request"
E_{10}	The root node receives a "get ESM-tree topology request" that is initiated by node N_i, and replies the ESM-tree topology (stores in $List_R$) information back to N_i.	"get ESM-tree topology request"	ESM-tree topology
E_{11}	The root node receives a "ping request" that is initiated by node N_i and replies an echo message back to N_i.	"ping request"	Echo message
E_{12}	The root node receives an "addition of node information request" that is initiated by node N_i. After this, $List_R$ is updated according to the information received.	"addition of node information request"	Nil
E_{13}	The root node receives a "removal of node information" request that is initiated by node N_i. After this, $List_R$ is updated according to the information received.	"removal of node information request"	Nil

Table 3
Description of State Transition Diagram in Figure 3 for different states and events.

Event	Description	Receive status of the root	Message sent by the root
E_{14}	The root node receives a "LP Lock request" that is initiated by node N_i and replies a "success" message back to N_i. After this, $LP_{R,i}$ is added to $L_{R,LP}$ and $\|L_{R,LP}\|$ is increased by 1. This implies that N_i may become a child of the root node.	"LP Lock request"	"success" message for the "LP Lock request"
E_{15}	The root node receives a "LP Lock request" that is initiated by node N_i and replies a "fail" message back to N_i. This is because $\|L_{R,LP}\| + NC_R < \mathcal{F}_R$.	"LP Lock request"	"fail" message for the "LP Lock request"
E_{16}	The root node receives a "free LP Lock request" that is initiated by node N_i. After this, $LP_{R,i}$ is removed from $L_{R,LP}$ and $\|L_{R,LP}\|$ is decreased by 1.	"free LP Lock request"	Nil
E_{100}	The root node receives a "connect as child request" from node N_j. This implies that $LP_{R,j}$ exists in $L_{R,LP}$. After this, N_j builds a real connection to the root node.	"connect as child request"	Nil

Table 4
Description of State Transition Diagram in Figure 3 for various events.

ure 4 are the state transition diagrams for a bootstrap server, a root node and a client node, respectively. In these state transition diagrams, we describe the states and the events for the *Tree Formation Protocol*. Table 2 is the explanation of Figure 2. Table 3 and Table 4 are the explanation of Figure 3. Table 5 and Table 6 are the explanation of Figure 4. The state transition diagrams of the other protocols will be shown in the later sessions.

In these state transition diagrams, events are made up by messages that are received

State	Description
C_0	This is the initial state of a client node, N_i. It has not connected to ESM service.
C_1	At this state, node N_i has just received a partial tree topology from the bootstrap server. It has not connected to ESM service.
C_2	This is the normal state of node N_i. It has connected to ESM service.
C_3	This is a "locked" state of node N_i. It is locked by one or more "LP Locks". (i.e. $\|L_{i,LP}\| \geq 1$). This implies that some nodes may become a children of node N_i.

Event	Description	Receive status of N_i	Message sent by N_i
E_{17}	The client node N_i, sends a "attach to ESM-tree request" to the bootstrap server and the bootstrap server replies a partial tree topology information back.	partial tree topology information	"attach to ESM-tree request"
E_{18}	N_i cannot connect to a node. It is because there is no registered ESM-tree.	"fail" message for the "attach to ESM-tree request"	"attach to ESM-tree request"

Table 5
Description of State Transition Diagram in Figure 4 for different states and events.

by or sent from a node. To illustrate these state transition diagrams for the "Tree Formation Protocol", let us consider a scenario in Figure **??** wherein an ESM-tree is formed initially. Initially, the bootstrap server is at state BS_0, the root node is at state R_0 and the client node N_i is at state C_0. The root node sends a "create ESM-tree request" to the bootstrap server and the bootstrap server replies a "success" message back to the root node. The corresponding events are E_7 in Figure 3 for the root node and E_1 in Figure 2 for the bootstrap server. Then the root node goes to state R_1 and the bootstrap server goes to state BS_1. Assuming node N_1 wants to join the ESM service. It first sends an "attach to ESM-tree request" to the bootstrap server. Then, the bootstrap server replies a partial tree topology information. The corresponding events are E_{17} in Figure 4 for node N_1 and E_4 in Figure 2 for the bootstrap server. Then the node N_1 goes to state C_1 and the bootstrap server remains in state BS_1. When N_1 receives the partial tree topology information, it tries to find a potential parent from this partial tree topology information. The potential parent of N_1 is the root node (as there is only a root node within the ESM-tree). N_1 sends a "LP Lock request" to the root node. If the root node replies a "success" message for this "LP Lock request", N_1 will send a "connect as child request" to the root node and make a real connection to the root node. Finally, a "free LP Lock request" will be sent by node N_1 to the root node. The corresponding events are E_{20} in Figure 4 for node N_1 and E_{14}, E_{100} and E_{16} in Figure 3 for the root node. After this, node N_1 goes to state C_2. The root node first goes R_1 to R_2 and then goes back to R_1.

3.2 **ESM: The Data Transfer Protocol**

In here, we focus on the general mechanism for implementing a reliable data transfer application such as file distribution. The data transfer process is initiated by the root node N_R which has a source data file F. The data transfer process consists of two phases, namely, (1) the *meta-data distribution* and, (2) the *data distribution*. For the meta-data distribution, N_R multicasts the meta informations about the file

Event	Description	Receive status of N_i	Message sent by N_i
E_{19}	N_i sends a "LP Lock request" to node N_j and N_j replies a "fail" message for the "LP Lock request". This implies that N_i wants to connect to N_j but N_j rejects N_i's request.	"fail" message for the "LP Lock request"	"LP Lock request"
E_{20}	N_i sends a "LP Lock request" to node N_j and N_j replies a "success" message for the "LP Lock request". After this, N_i sends a "connect as child request" to N_j and builds a real connection to N_j. Finally, N_i sends a "free LP Lock request" to N_j. This implies that N_i has successfully connected to the ESM-tree by choosing N_j as parent.	"success" message for the "LP Lock request"	"LP Lock request", "connect as child request" and "free LP Lock request"
E_{21}	N_i receives a "get ESM-tree topology request" that is initiated by node N_j and replies the ESM-tree topology information back to N_j.	"get ESM-tree topology request"	ESM-tree topology
E_{22}	N_i receives a "ping request" that is initiated by node N_j and replies an echo message back to N_j.	"ping request"	Echo message
E_{23}	N_i receives an "addition of node information request" that is initiated by node N_j. After this, $List_i$ is updated according to the information received.	"addition of node information request"	Nil
E_{24}	N_i receives a "removal of node information request" that is initiated by node N_j. After this, $List_i$ is updated according to the information received.	"removal of node information request"	Nil
E_{25}	N_i receives a "LP Lock request" that is initiated by node N_j and replies a "success" message back to N_j. After this, $LP_{i,j}$ is added to $L_{i,LP}$ and $\|L_{i,LP}\|$ is increased by 1.	"LP Lock request"	"success" message for the "LP Lock request"
E_{26}	N_i receives a "LP Lock request" that is initiated by node N_j and replies a "fail" message back to N_j. This is because $\|L_{i,LP}\| + NC_i < \mathcal{F}_i$ or $\|L_{i,LR}\| \neq 0$.	"LP Lock request"	"fail" message for the "LP Lock request"
E_{27}	N_i receives a "free LP Lock request" that is initiated by node N_j. After this, $LP_{i,j}$ is removed from $L_{i,LP}$ and $\|L_{i,LP}\|$ is decreased by 1.	"free LP Lock request"	Nil
E_{101}	N_i receives a "connect as child request" from node N_j. This implies that $LP_{i,j}$ exists in $L_{i,LP}$. After this, N_j builds a real connection to N_i.	"connect as child request"	Nil

Table 6
Description of State Transition Diagram in Figure 4 for various events.

F to all its children. These meta informations include (a) the file name of F, (b) the file size of F, and (c) the size of each data packet. For the data-distribution phase, the node N_R pushes the data packets to all its children also. Upon receiving a packet, each node forwards the received packet to its connected children nodes.

For the data transfer process, we have to consider the following issues. The first issue is that a new node, let say N_k, may join an ESM-tree while the data is being multicasted. Another issue is when an attached node N_k decides to perform a tree-optimization (which we will describe in the next section) and switches to another parent node. We handle these cases in the following manner. The node N_k needs to inform its parent node P_k: (i) the requested file name F, and (ii) its last received data packet $last\_received(N_k)$. If the node N_k is a newly joined node, $last\_received(N_k) = 0$. Two cases are considered: (1) The parent node P_k is still receiving the file F and $last\_received(P_k) \geq last\_received(N_k)$. In this case,

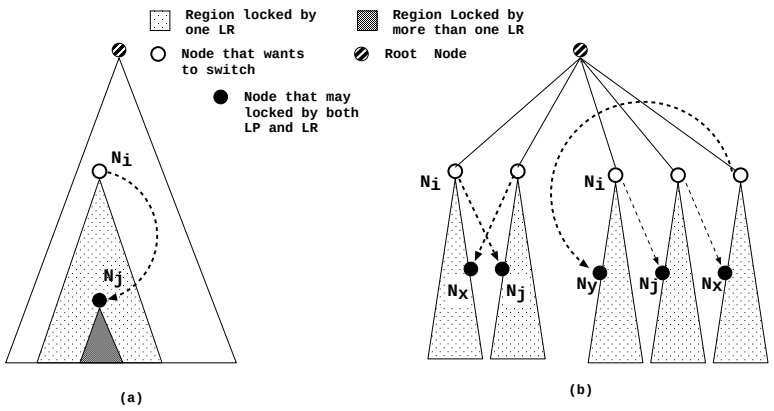

Fig. 5. Some examples for how cycles can be formed:(a) the node wants to switch to its descendant; (b) the generalize case

the node P_k can start the data transfer from data packet $last\_received(N_k) + 1$ to its child node N_k.(2) the parent node P_k is still receiving the same file F but $last\_received(P_k) < last\_received(N_k)$. In this case, the parent node P_k will not forward any data packet until it receives the data packet with the packet number equal to $last\_received(N_k) + 1$, then P_k can start the data transfer to node N_k.

3.3 **ESM: The Tree Optimization Protocol**

Tree optimization is to ensure that an ESM-tree can operate efficiently, such as good transfer bandwidth to all client nodes over a long period of time. We provide a distributed tree optimization protocol to ensure that the efficient operation of an ESM-tree and the ESM-tree can dynamically adapt to the changing network condition. The main idea about tree optimization is that each client node constantly monitors and probes[16] the transfer bandwidth with its parent node. If the transfer bandwidth drops below some threshold, then the client node will attempt to choose another parent node so that the client node and its descendant nodes can enjoy a high transfer bandwidth.

One important technical issue of tree optimization is on how to avoid tree partition or loop formation. Figure 5 illustrates this problem. Some nodes (those "unfilled" nodes) in Figure 5(a) & (b) attempt to perform a tree optimization and choose another potential parent node. If they choose any of their descendant nodes (e.g., as in Figure 5(a)), or they choose a node wherein its ancestor nodes are also in the process of performing tree optimization (e.g., as in Figure 5(b)), then tree partition and loop formation can occur. If this happens, those nodes that are not connected to the root node N_R will not be able to receive any data. Let us state the "necessary conditions" to partition an ESM-tree.

Necessary conditions to partition an ESM-tree:
Assuming a node N_i wants to choose another node N_j as its parent node. The

necessary conditions to partition an ESM-tree are:

- N_j is a descendant of N_i in an ESM-tree (e.g. as in Figure 5(a)), or
- N_i wants to switch to a node N_j, and an ancestor of N_j wants to switch to another node N_x, in which node N_x is a descendant node of N_i. (e.g. as in the left sub-tree in Figure 5(b)), or
- N_i wants to switch to a node N_j, and an ancestor of N_j wants to switch to another node N_x, and an ancestor of N_x wants to switch to node N_y, in which node N_y is a descendant node of N_i. Notice that this relation can be transitively propagated (e.g. as in the right sub-tree of Figure 5(b)).

To avoid the ESM-partition problem, we need to make sure that the above mentioned necessary conditions will not occurred. We propose a *"Distributed Locking Protocol"*. The main idea of this protocol is that for any node that wants to switch to another node, it prevents other nodes from finding its descendants as a new potential parent. Doing this can avoid the above mentioned necessary conditions and thereby eliminating loop formation or tree partition.

Assume that node N_i wants to perform a tree optimization operation, it needs to take (1) "LR Lock" on itself, (2) "LP Lock" on its potential parent, and (3) "LR Locks" for all nodes in T_i (sub-tree rooted by N_i). If any of the above locks cannot be taken, the whole procedure will be halted. The procedure executed by N_i.

Procedure tree_optimization(INPUT:$List_i$,OUTPUT:NULL) {
01 {
02 **if (bandwidth(BW) is below threshold)** {
03 /* Some nodes want to switch to N_i */
04 **if(** $|L_{i,LP}|$ **!=** 0 **)**
05 **exit;**
06 /* N_i or ancestors of N_i want to leave */
07 **if(** $|L_{i,LW}|$ **!=** 0 **)**
08 **exit;**
09 /* Lock itself to prevent other nodes from */
10 /* finding itself as a new parent */
11 /* Lock itself by $LR_{i,i}$ */
12 **Add** $LR_{i,i}$ **into** $|L_{i,LR}|$**;**
13 **Pick several nodes in** $List_i$ **and test the BW;**
14 $N_{PP} <-$ **the node that has the best BW;**
15 /* Lock parent to prevent parent's */
16 /* ancestors from conducting tree optimization */
17 **Lock** N_{PP} **by** $LP_{pp,i}$**;**
18 **if(fail to lock** $LP_{i,i}$ **)** {
19 **Free** $LR_{i,i}$**;**
20 **exit;**

```
21  }
22  /* Lock subtree to prevent other nodes from */
23  /* finding any node in N_i's subtree as a new parent*/
24  Lock Sub-Tree rooted at N_i with LR Locks
25  if( fail to lock Sub-Tree ) {
26    Free LR_{i,i};
27    Free LP_{pp,i};
28    Free LR locks in Sub-Tree;
29   }
30  Disconnect with old parent P_i;
31  Connect to new parent N_PP;
32  Free LR_{i,i};
33  Free LP_{pp,i};
34  Free LR locks in Sub-Tree;
35 }
36 }
```

If a node N_j receives a "LR Lock request" from N_i, it forwards this request to its children. N_j will reply a "success" message for the "LR Lock request" to N_i *only if* all children nodes of N_j reply "success" messages to N_j and both $L_{j,LP}$ and $L_{j,LW}$ are empty. The procedure executed by N_j when it receives a "LR Lock request" is:

```
Procedure reply_LR_request(INPUT:NULL,OUTPUT:NULL) {
01 {
02 /* Let N_x be an ancestor of N_j */
03 /* This LR lock is initiated by one of the ancestors*/
04 /* of N_j but it will forward to N_j only by P_{N_j} */
05 if N_j receives a lock LR_{j,x} request from its parent N_i {
06    /* someone wants to switch to N_j but N_j is locked by LP */
07    if ( |L_{j,LP}| != 0)
08       return "fail" to N_i;
09    if ( |L_{j,LW}| != 0)
10       return "fail" to N_i;
11    /* N_j is a leaf node */
12    if ( NC_j == 0) {
13       add LR_{j,x} into L_{j,LR}
14       return "success" to N_i;
15    }
16    /* forward this LR lock to all N_j's children */
17    for k from 0 to |List_j| - 1 {
18       if List_j[k] is N_j's children
19          send "LR_{j,x} request" to List_j[k];
20    }
21    wait for all children's replies;
```

```
22     if ( one or more than one of the children replies say "fail")
23        return "fail" to N_i;
24     else {
25        /* all nodes within the sub-tree are locked by LR*/
26        add LR_{j,x} into L_{j,LR}
27        return "success" to N_i;
28     }
29  }
30 }
```

If a node N_j receives a "LP Lock request" from N_i, it replies a "success" message for the "LP Lock request" *only if* both $L_{j,LR}$ and $L_{j,LW}$ are empty and N_j has not reached its fan-out limit. Here is the procedure executed by N_j when it receives a "LP Lock request" from N_i.

```
Procedure reply_LP_request(INPUT:NULL,OUTPUT:NULL) {
01  {
02  /*N_i wants N_j to become its new parent*/
03  if N_j receive a lock LP_{j,i} request from N_i {
04     /*Already locked by some ancestors */
05     if ( |L_{j,LR}| != 0)
06        return "fail" to N_i;
07     else if ( |L_{j,LW}| != 0)
08        return "fail" to N_i;
09     /* N_j had reach its Fan-out limit */
10     else if ( NC_j + |L_{j,LP}| >= F_j)
11        return "fail" to N_i;
12     else {
13        add LP_{j,i} into L_{j,LP};
14        return "success" to N_i;
15     }
16  }
17 }
```

We can show that the distributed locking protocol described above has the following property.

Theorem 3 *The distributed locking protocol described above avoids loop formation and tree partition.*

Proof: We can show this by contradiction. Assume that a cycle is formed during

the tree optimization. This implies that (a) the nodes that want to switch must lock its potential parent by "LP Locks" and all its descendants by "LR Locks", and (b) a node will not release the lock during the tree optimization operation. Also, no new node can join any node within the subtree of the switching node as all the descendants of the switching node are locked by "LR Lock"

Assume that a cycle is formed by a set of nodes $S = \{N_1, N_2, \ldots, N_i\}$, then every node in S must be both the descendant and ancestor of all other nodes within S (following the definition of a cycle). Since all the descendants of the switching node and the switching node itself must be locked by "LR Lock", all the nodes in S must be locked by "LR Locks". It is because there must be at least one potential parent node within the cycle. This implies that there must be at least one node that is simultaneously locked by both "LP Lock" and "LR Lock". However, this contradicts our specification. Therefore, no cycle can be formed during tree optimization procedure. ∎

3.3.1 State Transition Diagram for Tree Optimization Protocol

Figure 6 is the state transition diagram for a client node. It is an extension of Figure 4. In this state transition diagram, we describe the states and the events for the "Tree Optimization Protocol". Table 7 and Table 8 are the explanation of Figure 6. The state transition diagrams of other protocols will be shown in later sessions.

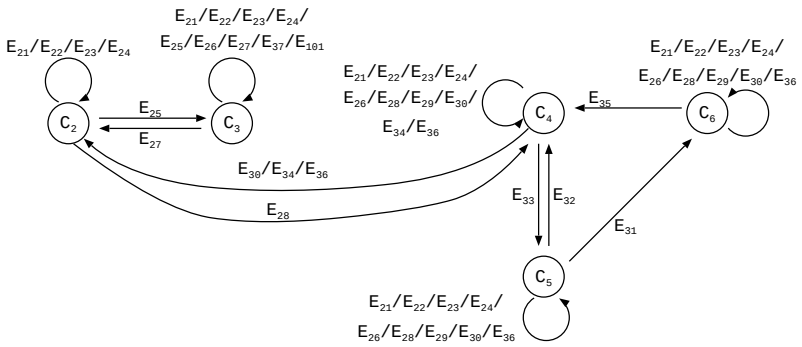

Fig. 6. State Transition Diagram of a Client Node for the Tree optimization Operation

To illustrate this state transition diagram for the "Tree Optimization Protocol", let us consider a scenario in Figure 7 wherein N_4 initiates the tree optimization procedure and N_4 wants to find N_1 as its new parent. At the beginning, node N_1, N_2, …, N_{10} are in state C_2. Assume that N_4 tries to add $LR_{4,4}$ into $L_{4,LR}$. If N_4 succeeds in adding this "LR Lock" on itself, it will forward the "LR Lock request" to its children nodes. The corresponding event is E_{28} in Figure 6 and N_4 goes to state C_4. N_4 sends a "LP Lock request" to its potential parent N_1. Assume that N_1 replies a "success" message back to N_4 for this "LP Lock request" and adds $LP_{1,4}$ into $L_{1,LP}$. The corresponding events are E_{33} in Figure 6 for N_4 and E_{25} in Figure 6 for N_1. After this, N_4 goes to state C_5 and N_1 goes to state C_3. When N_6 and

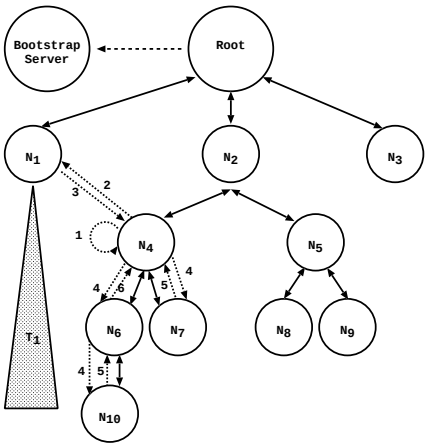

Fig. 7. ESM: Tree Optimization Protocol

State	Description
C_2	This is the normal state of a client node N_i. It has connected to an ESM service.
C_3	This is a "locked" state of node N_i. It is locked by one or more "LP Locks". (i.e. $\|L_{i,LP}\| \geq 1$). This implies that some nodes may want to become children of node N_i.
C_4	This is a "locked" state of node N_i. It is locked by one or more "LR Locks". (i.e. $\|L_{i,LR}\| \geq 1$). This implies that some ancestors of nodes N_i or N_i itself are attempting to switch to another parent.
C_5	This is a "locked" state of the client node. It is locked by one or more "LR Locks". N_i's potential parent is locked by "LP Lock". This is the second stage of "tree optimization" procedure. (a) N_i is locked by $LR_{i,i}$, and (b) its potential parent, N_j, is locked by $LP_{j,i}$.
C_6	This is the "locked" state of node N_i. This is the last stage of the "tree optimization" procedure. (a) N_i is locked by $LR_{i,i}$, (b) its potential parent, N_j, is locked by $LP_{j,i}$, and (c) T_i is locked by $LR_{x,i}$ (N_x is a node within T_i). At this state, N_i can switch to its potential parent.

Event	Description	Receive status of the root	Message sent by the root
E_{21} to E_{27}	Please refer to Table 6	Nil	Nil
E_{28}	N_i receives a "LR Lock request" that is initiated by node N_j (N_j may be N_i itself). N_i then forwards this message to all its children. After this, $LR_{i,j}$ is added to $L_{i,LR}$ and $\|L_{i,LR}\|$ is increased by 1. Notice that N_i will not reply a "success" or "fail" message for this "LR Lock" immediately.	"LR Lock request"	"LR Lock request"
E_{29}	N_i receives "success" messages for the "LR Lock request" from all its children within a time-out period. After that, N_i replies a "success" message to its parent for the "LR Lock" request.	"success" messages for the "LR Lock request"	"success" message for the "LR Lock request"

Table 7
Description of events of State Transition Diagram in Figure 6 for different states and events.

N_7 (children of N_4) receive the "LR Lock request" that is initiated by N_4, N_6 tries to add $LR_{6,4}$ into $L_{6,LR}$ and N_7 tries to add $LR_{7,4}$ into $L_{7,LR}$. Then, they forward this "LR Lock request" to their children (if any). The corresponding event is E_{28} in Figure 6 for both N_6 and N_7. After this, both N_6 and N_7 go to state C_4. Since N_7 has no children node, it replies a "success" message for the $LR_{7,4}$ lock immediately. The corresponding event is E_{29} in Figure 6 and N_7 remains in state C_4. Furthermore, when N_6 receives all its children's (i.e. N_{10}) "success" messages for

Event	Description	Receive status of N_i	Messages sent by N_i
E_{30}	N_i receives "fail" messages for the "LR Lock request" from some of its children, or N_i cannot receive any message from some of its children within a time-out period. After that, N_i replies a "fail" message to its parent.	"fail" messages for the "LR Lock request"	"fail" message for the "LR Lock request"
E_{31}	N_i receives "success" messages for the "LR Lock request" from all its children within a time-out period. This implies that N_i has locked T_i with "LR Lock" and N_i can go to next step in the tree optimization procedure.	"success" messages for the "LR Lock request"	Nil
E_{32}	N_i receives "fail" messages for the "LR Lock request" from some of its children, or N_i cannot receive any message from some of its children within a time-out period. This implies that N_i cannot lock T_i and must free up all the locks which N_i had taken before.	"fail" messages for the "LR Lock request"	"free LR Lock request" and "free LP Lock request"
E_{33}	N_i sends a "LP Lock request" to its potential parent N_j, and N_j replies a "success" message to N_i. This implies that N_j can accept N_i as its new child.	"success" message for the "LP Lock request"	"LP Lock request"
E_{34}	N_i sends a "LP Lock" request to its potential parent N_j, and N_j replies a "fail" message to N_i. This implies that N_j cannot accept N_i as its new child. N_i then free the $LR_{i,i}$ in $L_{i,LR}$.	"fail" message for the "LP Lock request"	"free LR Lock request"
E_{35}	N_i sends a "disconnect request" to its original parent and sends a "connect as child request" to its potential parent. After connected to the new parent, N_i sends a "free LP Lock" request to the new parent and broadcasts the new parent-child information.	Nil	"disconnect request", "connect as child request" and "free LP Lock request"
E_{36}	N_i receives a "free LR Lock request" that is initiated by node N_j. After this, $LR_{i,j}$ is removed from $L_{i,LR}$ and $\|L_{i,LR}\|$ is decreased by 1. Also, N_i will forward this message to its children (if any).	"free LR Lock request"	Nil
E_{37}	N_i receives a "LR Lock request" that is initiated by node N_j, through N_i's parent. N_i replies a "fail" message immediately to its parent as $L_{i,LP}$ is not empty.	"LR Lock request"	"fail" message for the "LR Lock request"

Table 8
Description of events of State Transition Diagram in Figure 6 for various events.

the "LR Lock request" from all its children node, N_6 replies a "success" message back to N_4 for the $LR_{6,4}$ lock. The corresponding events are E_{29} in Figure 6 for N_6 and E_{28} and E_{29} in Figure 6 for N_{10}. After this, both N_6 and N_{10} will in state C_4. Upon receiving all "success" messages for the "LR Lock request" from the children, N_4 knows that it has locked T_4. The corresponding event is E_{31} in Figure 6. To switch to a new parent node, N_4 goes to state C_6. N_4 then sends a "disconnect request" to N_2 and sends a "connect as child request" to N_1. Then, N_4 sends a "free LP Lock request" to N_1 and broadcasts the new parent-child pair information. The corresponding event is E_{35} in Figure 6 and N_4 goes to state C_4. After N_1 received the "connect as child request" and "free LP Lock request" from N_4 , N_4 becomes a child of N_1. The corresponding events are E_{27} and E_{101} in Figure 6 for N_1. and N_1 goes back to state C_2. Later, N_4 sends a "free LR Lock request" to itself. As a result, $LR_{4,4}$ is removed from $L_{4,LR}$ and N_4 forwards the "free LR Lock request" to its children (if any). The corresponding event is E_{36} in Figure 6. and N_4 goes back to state C_2. Finally, N_6 and N_7 receive a "free LR Lock request" from N_4. This causes the removal of $LR_{6,4}$ from $L_{6,LR}$ in N_6 and the removal of $LR_{7,4}$ from $L_{7,LR}$ in N_7. Also, they forward this request to theirs children (if any). The corre-

sponding event is E_{36} in Figure 6 for both N_6 and N_7. After this, both N_6 and N_7 go back to state C_2.

3.4 ESM: The Node Leaving Protocol

A node may want to leave an ESM-tree at any time and may forward data to its children. If a node wants to leave a tree, the sub-tree under it will be partitioned from the original ESM-tree. Thus, special procedures are needed to handle this node leaving event. The main technical difficulty in handling the node leaving event is similar to that of the tree partition problem in the "tree optimization" procedure. The difference is that a sub-tree under a leaving node *must* switch to another parent, while in the "Tree Optimization" operation, the procedure will be halted and be restored the node cannot successfully take all the required locks from its descendent nodes. The leaving node's children must wait until all necessary locks have been taken successfully before they switch to another parent node.

The main idea of the node leaving protocol is that when the node N_i leaves, the sub-tree T_i (e.g. the subtree where N_i is the root node) will be locked by "LW Locks" which are initiated by N_i. For those nodes that are locked by "LW Locks", they will reject any new coming "LP" and "LR" locking request. At this time, other nodes may be in the process of leaving/joining the sub-tree T_i. After T_i is locked by "LW Lock", *only* the children of N_i will perform the node switching event.

Assume that a node N_i wants to leave. We divide this node leaving operation into three components. They are : (1) procedure for the leaving node N_i, (2) procedure for those nodes that are not within T_i and, (3) procedure for those nodes that are within T_i The procedure that node N_i needs to perform when it leaves is:

Procedure leave(INPUT:NULL,OUTPUT:NULL) {
01 {
02 N_i **locks itself by** $LW_{i,i}$
03 **/* wait for other tree optimization processes to be finished */**
04 **if(** $L_{i,LP} \neq$ **empty or** $L_{i,LR} \neq$ **empty** }
05 **wait for both** $L_{i,LP}$ **and** $L_{i,LR}$ **become empty**
06 }
07 **ESM-Broadcast the node leave event**
08 **Leaves the tree**
09 }

For a node N_j that is not within T_i, it will delete the information of T_i in $List_j$. Here is the procedure that N_j needs to response to N_i's leaving message.

Procedure other_node_leave

(INPUT: address of leave node(N_i), OUTPUT:$List_j$) {
01 {
02 N_j **deletes ALL tree information for** T_i
03 if($LP_{j,x}$ **exists in** $L_{j,LP}$ **where** N_x **is within** T_i **)** {
04 **/*** N_x **is switching to** N_j ***/**
05 N_j **waits for** N_x **to complete the switching procedure**
06 **/* As** N_j **delete the information before */**
07 **synchronize** $List_j$ **and** $List_x$
08 }
09 /*broadcast N_i**'s failure */**
10 N_j **update status via flooding**
11 }

For a node N_k that is within T_i, it will be locked by $LW_{k,i}$. Then, it will forward N_i's leaving information to its children (which will cause N_k's children to be locked by "LW Locks"). After $L_{k,LP}$ and $L_{k,LR}$ become empty, N_k waits for the replies from its children. When all of the N_k's children reply "success" messages to N_k, N_k replies a "success" message for the "LW Lock" to its parent. Finally, only the children of the leaving node (i.e. N_i's children) will find a new parent within the ESM-tree. Here is the procedure that node N_k needs to response to N_i's leaving message.

Procedure ancestors_leave
(INPUT: address of leave node(N_i), OUTPUT:$List_j$) {
01 {
02 N_k **locks itself by** $LW_{k,i}$
03 /* If N_k **has children,** N_k **needs to forward the event to them */**
04 if($NC_k \neq 0$ **)** {
05 **Forward "LW Lock" request to children**
06 }
07 while($L_{k,LP}$ **is NOT empty)** {
08 **/* some nodes may become children of** N_k ***/**
09 **Wait for the switching procedure complete**
10 **Forward "LW Lock" request to the new children**
11 }
12
13 while($L_{k,LR}$ **is NOT empty)** {
14 **/*** N_k **or** N_k**'s ancestors want to switch */**
15 if($LR_{k,k}$ **exists in** $L_{k,LR}$ **)** {
16 **/*** N_k **wants to switch */**
17 **continue switching procedure**
18 **if(switching is success)** {
19 **Free "LR Lock" in** T_k

```
20        Free "LW Lock" in T_k
21      }
22      else if( switching is NOT success) {
23        Free "LR Lock" in T_k
24      }
25    }
26  }
27
28  /* If N_k has children, N_k needs to lock T_k before it can do a switching process */
29  if( NC_k ≠ 0 ) {
30    wait for all children's replies for
31      the successfully taking of "LW Lock"
32  }
33
34  /* In here, all the children nodes have "LW Lock" */
35  /* This implies all the children */
36  /* have finished the switching procedure */
37  if( P_k ≠ N_i ) {
38    /* Replies success of taking "LW Lock" to parent */
39    reply "success" to P_k
40  }
41  else if( P_k == N_i) {
42    /* N_k is N_i's(the leaving node) children */
43    /* Switch to a new parent */
44    /* Note : For this switch, N_k no need to lock T_k with "LR Lock" */
45    /* as T_k is already locked by "LW Lock" */
46    /* Only "LP Lock" at the potential parent node is needed. */
47    switch to a node that is not in the T_i
48    Syn. List_k and List_{P_k}
49    Send free "LW Lock" message to children
50  }
51 }
```

3.4.1 State Transition Diagram for Node Leaving Protocol

Figure 8 is the state transition diagram for a client node. It is an extension of Figure 4 and Figure 6. In this state transition diagram, we describe the states and the events for the "Node Leaving Protocol". Table 9 and Table 10 are the explanation of Figure 8.

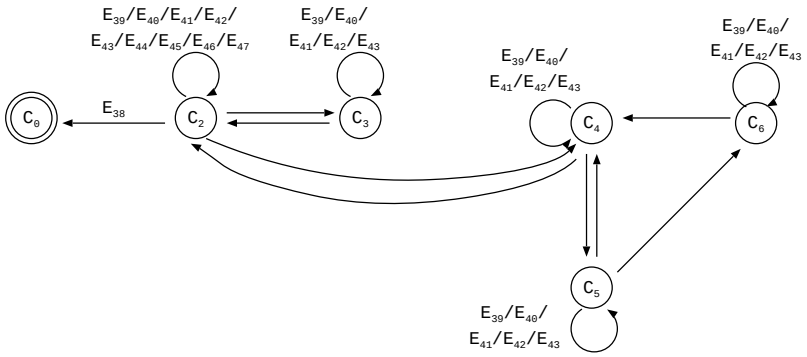

Fig. 8. State Transition Diagram of a Client node for Node Leaving Protocol

State	Description
C_0	This is the initial and final state of a client node N_i (It is also the initial state of a client node). At this state, N_i has left the ESM service.
C_2 to C_6	Please refer to Table 7

Event	Description	Receive status of N_i	Messages sent by N_i
E_{38}	N_i broadcasts the "leave request" and leaves the ESM-tree. This implies that both $L_{i,LP}$ and $L_{i,LR}$ are empty.	Nil	"leave request"
E_{39}	N_i receives a "LW Lock request" from itself. After receiving this message, $LW_{i,i}$ is added into $L_{i,LW}$. This implies that N_i wants to leave.	"LW Lock request"	Nil
E_{40}	N_i receives a "leave request" from N_j. If N_j is the parent of N_i, N_i will add $LW_{i,j}$ into $L_{i,LW}$ and forward a "LW Lock request" to its children (if any). If N_j is NOT the parent of N_i, T_j will be deleted from $List_i$.	"leave request"	"LW Lock request" (with condition)
E_{41}	N_i receives a "LP Lock request" from N_j and replies a "fail" message to N_j as $L_{i,LW}$ is not empty.	"LP Lock request"	"fail" message for the "LP Lock request"
E_{42}	N_i receives a "LR Lock request" from N_j and replies a "fail" message to N_j as $L_{i,LW}$ is not empty.	"LR Lock request"	"fail" message for the "LP Lock request"
E_{43}	N_i receives a "LW Lock request" that is initiated by N_j (that means N_j is the leaving node) through N_i's parent. After receiving this message, $LW_{i,j}$ is added into $L_{i,LW}$ and N_i forwards this "LW Lock request" to its children (if any). This implies that N_i's ancestor (i.e. N_j) wants to leave.	"LW Lock request"	"LW Lock request" (with condition)

Table 9
Description of State Transition Diagram in Figure 8 for different states and events.

3.5 **ESM: The Node Failure Protocol**

A node N_i may disconnect from the ESM-tree at any time due to node failure. For this type of failure, it is not possible for node N_i to inform other nodes of this failure event. Thus, special procedures are needed to handle this node failure event. In general, a node N_j can detect the failure of node N_i by the following events:

- When node N_j sends a request message to node N_i and does not get any reply

Event	Description	Receive status of N_i	Messages sent by N_i
E_{44}	N_i has no children and P_i is NOT the leaving node. After this, N_i sends a "success" message back to its parent for the "LW Lock".	Nil	"success" message for the "LW Lock"
E_{45}	N_i has no children and P_i is the leaving node. N_i sends a "LP Lock request" to a node that is not within T_i. If the node replies a "success" message, then N_i can switch to the new parent. Otherwise, N_i keeps sending "LP Lock request" to different nodes that are not within T_i. (Note: This switching does not need "LR Lock request" as N_i is already locked by "LW Lock")	Nil	"LP Lock request"
E_{46}	N_i receives all its children's "success" messages for the "LW Lock" and P_i is NOT the leaving node. After this, N_i replies a "success" message to P_i.	"success" message for the "LW Lock"	"success" message for the "LW Lock" (with condition)
E_{47}	N_i receives all its children's "success" messages for the "LW Lock" and P_i is the leaving node. N_i sends a "LP Lock request" to a node that is not within T_i. If the node replies a "success" message, then N_i can switch to the new parent. Otherwise, N_i keeps sending "LP Lock request" to different nodes that are not within T_i. (Note: This switching does not need "LR Lock request" as T_i is already locked by "LW Lock")	"success" message for the "LW Lock"	"LP Lock request"

Table 10
Description of State Transition Diagram in Figure 8 for various events.

from node N_i within a time-out limit, node N_j will consider node N_i has failed.

- Node N_j can assume its neighboring node N_i has failed if node N_j does not receive any request from node N_i after, a time-out limit. The implies that every nodes need to send a "alive" message to their neighbors if there is no communication for a while.

The main idea of the "Node Failure Protocol" is that when a node N_j finds that node N_i has failed, it will notify the other nodes for this failure event. Then, all nodes will handle N_i failure by the "Node Leaving Protocol". Here is the procedure that node N_j needs to response to N_i's failure message.

Procedure node_fail(INPUT:address of the fail node(N_i),OUTPUT:$List_j$)
01 {
02 /* ignore the message if N_i is already removed from $List_j$ */
03 if (N_i does not exist in $List_j$) return;
04 /* use "Node Leaving Protocol" to handle N_i's failure */
05 if (N_i is an ancestors of N_j)
06 ancestors_leave(N_i, $List_j$);
07 else
08 other_node_leave(N_i, $List_j$);
09 /*broadcasts N_i's failure */
10 N_j updates status via flooding;
11 }

---

4 Performance Evaluation

In this section, we present experimental results to illustrate the soundness and effectiveness of our proposed ESM service. The performance measure that we are interested in is the completion time of file distribution under our ESM architecture. For the first three experiments, we use our ESM prototype to compare with different unicast approaches. We also investigate the performance of ESM under different network conditions (e.g., with or without background traffic) as well as the improvement of file distribution completion time under the tree optimization operation. For the last experiment, we use the packet-level simulator NS2 to study the performance of the ESM architecture in a large-scale network.

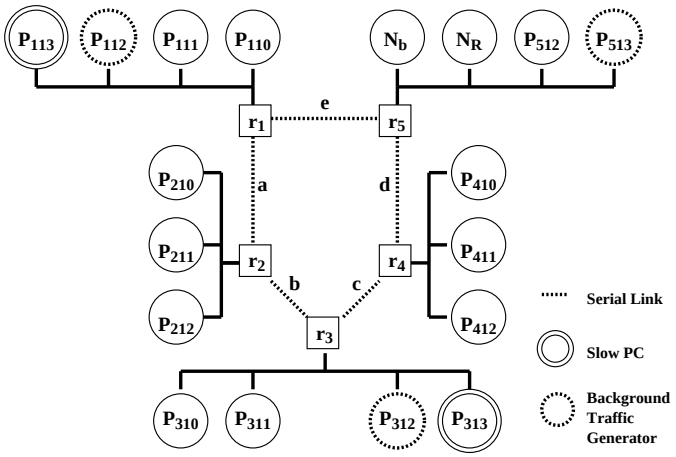

Fig. 9. Experimental Setup

Figure 9 illustrates our experimental setup for the first three experiments. There are 18 computing nodes, running at five different network domains. One of the computing nodes is the root node N_R. There are five routers in the experimental setup, r_1 to r_5. Unless we state otherwise, the links between the routers (link a,b,c,d and e) have a transfer bandwidth of 4Mbps. All other links in the system have a bandwidth of 100 Mbps. For the 18 computing nodes, two of them are of lower configuration (e.g., AMD K6-300 with 16 MB memory so as to model hand held device) and they are P_{113} and P_{313}. The other computers are have a minimum of 128 MB memory. Three computing nodes are used to generate background traffics and they are P_{112}, P_{312}, and PC_{513}. All transfer sessions are carried out using TCP.

4.1 Experiment 1 - Comparisons between IP Unicast and the ESM prototype

In experiment 1, we record the finishing time of a reliable file transfer. The size of the file is $50MB$. We carry out the experiment under two settings.

- **setting A :** there is no background traffic in the network.
- **setting B :** there are three TCP cross traffics inside the network.

The three TCP traffics are: (1) from P_{112} to P_{312} through r_1, r_2 and r_3, and (2) from P_{312} to P_{512} through r_3, r_4 and r_5, and (3) from P_{513} to P_{112} through r_1 and r_5.

We consider three cases in these experiment, they are:
Case 1 : IP unicast, single file transfer – In this case, the root node, N_R, transfers the file to a specific node one at a time. The target nodes are P_{110}, P_{111}, P_{113}, P_{210}, P_{211}, P_{212}, P_{310}, P_{311}, P_{313}, P_{410}, P_{411}, P_{412} and P_{512}. Note that this is the ideal file completion time for N_R to transfer the file to that specific node.
Case 2 : IP unicast, multiple file transfer – In this case, N_R transfers the file to all the client nodes P_{110}, P_{111}, P_{113}, P_{210}, P_{211}, P_{212}, P_{310}, P_{311}, P_{313}, P_{410}, P_{411}, P_{412} and P_{512} at the same time. This implies that N_R starts 13 TCP sessions concurrently. We investigate the situation wherein the root node N_R or the links may become the bottleneck. Note that this is indeed the common scenario for multiple file transfer on the Internet.
Case 3 : ESM, multiple file transfer – In this case, we record the completion time for transferring the file by the ESM-tree topology G_1. The graph G_1 has the following topology: N_R is the parent node of P_{110}, P_{210}, P_{310}, P_{411} and P_{512}; P_{110} is the parent node of P_{111} and P_{113}; P_{210} is the parent node of P_{211} and P_{212}; P_{310} is the parent node of P_{311} and P_{313}; P_{411} is the parent node of P_{410} and P_{412}. The data transfer process is the same as described in the *Data Transfer* part of section 3.

	Case1		Case2		Case3	
	A	**B**	**A**	**B**	**A**	**B**
P_{512}	29.08	30.18	29.16	31.55	29.15	30.11
P_{412}	160.12	331.26	847.55	1025.67	309.23	658.36
P_{411}	159.36	330.45	856.19	1030.98	308.15	657.92
P_{410}	158.09	333.23	852.34	1031.33	310.22	658.23
P_{313}	205.36	404.68	942.56	1362.21	378.56	741.95
P_{311}	168.92	364.22	881.26	1283.04	331.25	703.36
P_{310}	169.02	360.24	876.45	1278.67	329.65	702.35
P_{212}	170.22	364.66	881.90	1305.23	326.23	706.10
P_{211}	169.01	365.83	879.45	1299.04	325.06	706.32
P_{210}	170.26	367.89	886.99	1301.45	324.25	705.26
P_{113}	192.04	395.67	920.73	1109.93	341.23	691.23
P_{111}	160.55	334.98	854.34	1037.98	311.98	652.36
P_{110}	159.12	331.23	856.35	1051.08	311.65	653.01

Table 11
file transfer time (in unit of second) for Experiment 1

Summary for Experiment 1: Table 11 illustrates the result. We observe that :

- Case 1 is the optimal file transfer time. Comparing with Case 3 under setting A, the results are comparable and ESM only runs slightly worse than the ideal situation (Case 1). For setting B, ESM takes a bit longer to complete the transfer because it is transferring the file to multiple nodes at the same time.

- In Case 2, N_R uses IP unicast (via TCP) to transfer the file to all nodes at same time. Comparing with Case 3 of the ESM file transfer, for both settings, the results show that ESM performs much better. For setting B, the improvement of file transfer times to P_{310} and P_{311} is much greater than that of P_{110} and P_{111}. This is because the data packets need to pass through routers r_3, r_4 and r_5 to reach P_{310} and P_{311}, whereas the data packets need to pass through routers r_1 and r_5 only to reach P_{110} and P_{111}. The more routers the data packets need to pass through, the higher the chance that they may be lost.

The result shows that ESM generally performs better than IP unicast when we want to transfer data to multiple clients at the same time. Another important point is that the performance of the ESM server is "topology" dependent. We explore this issue in the next experiment.

4.2 Experiment 2 - Comparisons between different ESM topologies

In experiment 2, the setup is similar to that of experiment 1. All data transfer process is the same as described in the *Data Transfer* part of section 3. We perform the experiment with five cases and they are:

Case 1 : ESM, topology G_2 **–** In this case, we record the file completion times for transferring the file by ESM-tree topology G_2. The graph G_2 has the following topology: N_R is the parent node of P_{512}, P_{210} and P_{310}; P_{310} is the parent node of P_{311} and P_{313}; P_{210} is the parent node of P_{110} and P_{212}; P_{212} is the parent node of P_{211}; P_{110} is the parent node of P_{111} and P_{113}; P_{311} is the parent node of P_{411}; P_{411} is the parent node of P_{410} and P_{412}.
Case 2 : ESM, topology G_3 **–** In this case, we record the file completion times for transferring the file by ESM-tree topology G_3. The graph G_3 has the following topology: N_R is the parent node of P_{512}, P_{110} and P_{411}; P_{110} is the parent node of P_{111} and P_{113}; P_{411} is the parent node of P_{410} and P_{412}; P_{410} is the parent node of P_{310}; P_{310} is the parent node of P_{311} and P_{313}; P_{311} is the parent node of P_{210}; P_{210} is the parent node of P_{212}; P_{212} is the parent node of P_{211}.
Case 3 : ESM, topology G_3 **with a slow link "e" –** In this case, we record the file completion times for transferring the file by a tree topology that is the same as the one in Case 2 (G_3). The difference is that the link speed between r_1 and r_5 (link e) is configured to 56 kbps.
Case 4 : ESM, topology G_4 **–** The graph G_4 has the following topology: N_R is the parent node of P_{512} and P_{411}; P_{411} is the parent node of P_{410} and P_{412}; P_{410} is the parent node of P_{310}; P_{310} is the parent node of P_{311} and P_{313}; P_{311} is the parent node of P_{210}; P_{210} is the parent node of P_{212}; P_{212} is the parent node of P_{211}; P_{210} is the parent node of P_{110}; P_{110} is the parent node of P_{111} and P_{113}.
Case 5 : ESM, topology G_3 **with a tree optimization operation –** In this case, we record the file completion times for transferring the file when a tree optimization operation is performed. At the beginning, the configuration is same as Case

3. However, node P_{110} discovers that the link performance between its parent(N_R) and itself is not good enough. Then, it performs a tree optimization operation and finds P_{210} as its new parent. Finally, the tree topology becomes G_4.

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**
P_{512}	29.10	29.08	29.17	29.36	30.29
P_{412}	370.26	184.96	193.95	169.44	176.02
P_{411}	364.26	184.58	192.36	169.36	175.96
P_{410}	365.23	185.36	193.65	169.90	175.99
P_{313}	382.32	236.23	240.11	235.23	238.25
P_{311}	359.42	197.02	198.63	202.66	205.36
P_{310}	359.38	196.42	197.36	202.60	205.61
P_{212}	340.30	209.93	213.28	209.02	210.25
P_{211}	343.33	211.23	215.45	210.35	211.26
P_{210}	340.28	208.26	210.01	208.72	209.99
P_{113}	399.32	222.36	7713.01	249.23	246.23
P_{111}	353.00	178.06	7691.26	212.00	219.96
P_{110}	352.48	178.04	7689.82	211.98	218.16

Table 12
file transfer time (in unit of second) for Experiment 2 in Setting A (without background traffic)

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**
P_{512}	29.45	29.34	29.99	31.86	31.89
P_{412}	645.96	390.26	396.45	345.23	350.95
P_{411}	642.23	388.26	396.10	343.55	348.96
P_{410}	643.26	381.36	397.65	344.85	351.21
P_{313}	682.26	469.73	496.36	447.89	440.51
P_{311}	635.23	436.95	432.69	408.91	401.36
P_{310}	633.26	434.23	431.26	409.27	401.01
P_{212}	620.91	441.36	450.36	415.26	423.25
P_{211}	621.54	445.69	452.33	416.23	424.12
P_{210}	620.49	440.36	449.23	412.36	423.14
P_{113}	672.10	415.69	8000+	464.69	471.23
P_{111}	633.26	378.75	8000+	423.29	434.26
P_{110}	631.69	376.95	8000+	422.81	433.27

Table 13
file transfer time (in unit of second) for Experiment 2 in Setting B (with TCP background traffic)

Summary for Experiment 2: Table 12 and Table 13 and illustrates the result of experiment 2. We observe that:

- Case 1, Case 2 and Case 4 are ESM with different tree topologies. By comparing their results, we observe that the performance of ESM is indeed “topology”

dependent. For example, the completion time of setting B in Case 2 differs significantly from that of Case 1 and Case 4.

- Case 2 and Case 3 share the same topology, G_3. The only difference is that in Case 3, the bandwidth of link "e" is reduced to 56 kbps. As we can see when there is no tree optimization, P_{110}, P_{111} and P_{113} have a poor performance. Case 5 is the result when a tree optimization is performed when P_{110} switches to a new parent and changes the tree topology to G_4. From the result, we observe that tree optimization can help a node to find a better parent and to receive data at a faster rate.
- Case 4 and Case 5 share the same topology, G_4. The only difference is that in Case 5, there is one tree optimization performed. From both settings A and B, it shows that tree optimization will only slightly increase the transfer time.

The result shows that ESM performance depends on the tree topology. Also, the tree optimization procedure is an important protocol for the ESM to improve the performance of data transfer. As the link conditions between nodes are changing all the times, the nodes and their sub-trees may suffer a lot. This is the justification of the necessity of the tree optimization protocol for the ESM service.

4.3 Experiment 3 - Comparison between different thresholds for tree optimization operation in our ESM prototype

In this experiment, we focus on the completion time of a specify node, P_{110}. We transfer a $50MB$ file. The tree topology at the beginning is G_3. There is no background traffic in the network at the beginning.

After 2 seconds, the root starts the file transfer. P_{513} starts to generate a background traffic to P_{112}. The cross traffic will consume some of the bandwidth of link "e". This cross traffic will cause P_{110} to perform tree optimization operation and to switch to a better parent, P_{210}. The tree topology will be changed to G_4 afterward. We carry out the experiment under two settings. For setting A, the cross traffic is UDP traffic. For setting B, the cross traffic is TCP traffic.

For the tree optimization operation, each node keeps track of two variables. They are:

- $CUR(i) =$ "current transfer rate for the ith packet"
- $AVG(i) =$ "average rate for the i packet", which is calculated as:

$$AVG(i) = (1 - a) * AVG(i - 1) + a * CUR(i)$$

where $AVG(0) = 0$ and $a = 0.1$. Table 3 illustrates the result of Exp. 3 wherein

- "ESM (G_3)" represents the completion time for transferring the file to P_{110} for tree topology G_3 under no cross traffic situation.

- "ESM (G_4)" represents the completion time for transferring the file to P_{110} for tree topology G_4 under no cross traffic situation.
- "X" represents the threshold for the node, P_{110}, to start the tree optimization switching process. The switching will take place when $CUR(i) \leq X * AVG(i)$.

	A (UDP backgroud traffic)	**B (TCP background traffic)**
ESM (G_3)	178.04	178.04
ESM (G_4)	211.98	211.98
$X = 0.1$	420.23	342.22
$X = 0.2$	419.36	344.08
$X = 0.3$	417.23	343.26
$X = 0.4$	418.22	345.76
$X = 0.5$	420.45	342.36
$X = 0.6$	419.32	218.26
$X = 0.7$	218.36	219.23
$X = 0.8$	236.89	231.78
$X = 0.9$	275.87	268.45
$X = 1.0$	435.23	430.25

Table 14
file transfer time for P_{110} (in unit of second) for Experiment 3

Summary for Experiment 3: we observed that

- By comparing the results for $X = 0.5$ to 1.0 in Table 14, it suggests that we should not set the value of X too high (e.g., $X = 1.0$). The reason is that if the value of X is too high, then node tries to perform tree optimization very often even when there is little fluctuation in the transfer bandwidth. The more often a node tries to switch to a new parent node, the longer it takes to finish the file transfer. The result from Table 14 also suggests that there is an optimal value for the activation threshold X so as to minimize the file completion time.

4.4 Experiment 4 - NS2 Simulation for Large Scale Network

In experiment 4, we carry out a large scale packet level simulations in NS2[31]. The performance measure that we are interested in is the completion time of file distribution. The size of file is 5MB. We compare our ESM architecture with unicast. We also investigate the performance under different network conditions (e.g., with or without background traffic, with different number of clients in an ESM-tree).

We simulate our ESM architecture with 10, 20, 30, 40, 50, 100, 150, 200 and 300 nodes topologies. In each topology, we partition the network into 5 domains. Each domain is connected to two other domains and these domains form a cycle. Links between domains have 1Mbps bandwidth. Links within each domain have a transfer bandwidth between 3 to 100 Mbps, which are evenly distributed. An example of 100-node topology is shown in Figure 10. In each topology, we carry out six simulations:

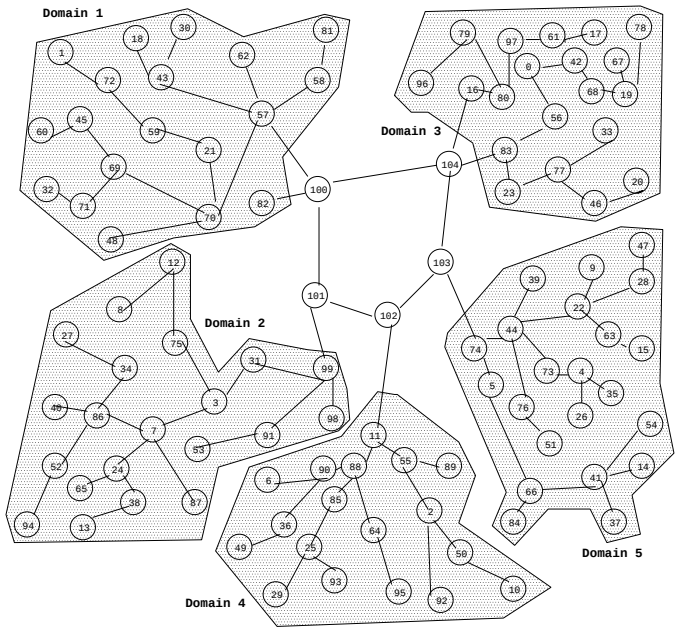

Fig. 10. 100-Node Topology

IP Unicast – In this case, N_R tries to send the file to all client nodes by IP Unicast at the same time. The resulting time is the average of the completion time (the time between N_R starts the transfer and a client node completely receives the file) of each node.
Ideal – In this case, N_R tries to send the file to all the client nodes by IP Unicast. Moreover, N_R will send the file to the client nodes one by one. The resulting time is the average of the completion time of each node.
ESM – In this case, N_R tries to send the file to all the client nodes using the ESM protocol. The resulting time is the average of the completion time of each node.
IP Unicast w/UDP cross traffic – In this case, all settings are the same as the case with "IP Unicast" except there are 5 Constant Bit Rate (CBR) UDP cross traffics. Each CBR is occupying one cross-domain link with a traffic rate of 0.5Mbps.
Ideal w/UDP cross traffic – In this case, all settings are the same as the case with "Ideal" except there are 5 CBR UDP cross traffics. Each CBR is occupying one cross-domain link with a traffic rate of 0.5Mbps.
ESM w/UDP cross traffic – In this case, all settings are the same as the case with "ESM" except there are 5 CBR UDP cross traffics. Each CBR is occupying one cross-domain link with a traffic rate of 0.5Mbps.

Summary for Experiment 4: The results of experiment 4 are shown in Figure 11.

- By comparing the "IP Unicast" scheme and the "ESM" scheme in both with and without background traffic, we can conclude that the "ESM" scheme has a much shorter file completion time, as compare with the unicast in all cases. This also shows the effectiveness of the "ESM" scheme in a large scale network.
- By comparing the "Ideal" scheme and the "ESM" scheme in both with and with-

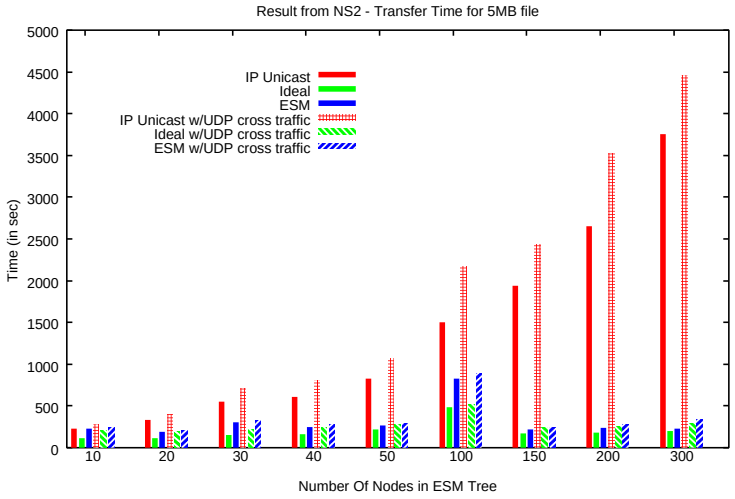

Fig. 11. File Transfer Time (in unit of second) in NS2

out background traffic, we can conclude that the "ESM" scheme runs slightly worse than the ideal situation in a small scale network. When the scale of the network becomes large, the performance of the "ESM" scheme can be comparable with the "Ideal" scheme (with respect to the "IP Unicast" scheme).

5 Related Work

In this section, we describe some of the related work in ESM. ALMI[33] is an application level infrastructure to provide multicast services to the end system. It uses a *centralized* approach to maintain the multicast tree. Only the "session controller" handles the members joining and maintains the multicast tree. Members measure the distance among them and send this information to the "session controller". The "session controller" computes the multicast tree by finding a MST. Data is transferred along the multicast tree, while control messages are transferred by using unicast with each member. The main difference between ALMI and our ESM is that ALMI is an centralized approach while our ESM is a distributed approach to maintain the multicast tree.

Banana Tree Protocol(BTP)[21] is designed for a file sharing program, Jungle Monkey[29]. It assumes the existence of some bootstrap protocols to handle members joining. Nodes in BTP can change their parents. To prevent a partitioning of the multicast tree, BTP restricts the potential parent of a switching process. The potential parent of a switching process (1) must be a sibling of the switching node, and (2) must not attempt to switch to another parent. The main difference between BTP and our ESM is that in BTP, a switching node can only switch to its siblings while our is a more general approach for tree optimization that can avoid deadlock, loop formation and tree partition.

Narada[10] is a protocol focusing on the efficiency of the overlay structure. The

multicast tree is created from a mesh by Narada's enhance distance vector routing strategy. A node can join the services by a bootstrap procedure. Each member stores a *list* of others members, and constantly probes the other members in the *list*. Narada relies on this probing to maintain connectivity for the mesh. When a node leaves, it notifies the other members to delete itself in others' list. Tree partition is detected by timeout ("refresh" message) in Narada. The main difference between Narada and our ESM is that in Narada uses the partitition detection approach while our is a partition avoidance approach.

Bayeux[44] is an application infrastructure for end hosts multicast. It is based on consistent hashing functions used in the Chord and Tapestry[36,43]. In Bayeux, there is a set of nodes (called "root") to handle the multicast tree maintenance such as tree creation, node joining and node leaving. Also, nodes will not change their "root" after they joined the service. Bayeux depends on the "Explicit Knowledge Path Selection" protocol to periodically update the routing tables in order to select a better data delivery path. The main difference between Bayeux and our ESM is that in Bayeux, a node will not change their parent after they joined the service. On the other hand, our ESM allows nodes to switch to a better parent node so that the node and its associated sibling nodes will receive better quality-of-service.

Host Multicast[41] is a hybrid framework of IP unicast and IP multicast. For nodes that are capable to communicate by using IP multicast, they use IP multicast. Otherwise, they use IP unicast to communicate. For each node, it runs a daemon process in user space to provide end system multicast functions. The bootstrap and joining procedure is systematic and hierarchical. Fail node is detected by timeouts ("REFRESH" message). Nodes can change their parents if they find a better one. To avoid loop formation, members will detect themselves whether they are within a loop or not (by exchange of "PATH" message). If a loop is formed, one of the member within the loop will detect the loop. The main difference in Host Multicast and our ESM is that Host Multicast uses a loop detection mechanism while our is a loop avoidance mechanism.

Like Narada, Scattercast[8] also takes the mesh-based approach. The multicast tree is formed from the mesh that is connecting different nodes. The cost evaluation functions of Narada and Scattercast are different. The main difference between Scattercast and Narada is that Scattercast will co-operate some proxy-like agents (called "SCX") in its structure. These SCXs will handle most of the multicast functions such as mesh optimization and node leaving. The end-system only needs to join one of the SCXs to enjoy the multicast services. The main difference between Scattercast and our ESM is that we allow node to switch to other nodes so as to receive better service.

6 Conclusion

In this paper, we propose an architectural framework for performing an ESM service. One advantage of ESM is that it resolves the deployment problems of IP multicast. To have a high ESM service performance, one has to carefully design various protocols so as to make this distributed service correct and consistent. We propose and implement the distributed protocols for the tree formation, date transfer, tree optimization, node leaving and node failure events for the ESM service. We prove the correctness and properties of these procedures, for example, we can maintain a tree topology after clients joining event or a tree optimization operation and that no tree partition can occur in an ESM-tree. We carried experiments to illustrate the soundness and the effectiveness of the ESM service. We show that ESM can have a comparable performance even when compare with the ideal condition for data transfer. Our work provides an architectural framework for people to deploy multicast service.

References

[1] Suman Banerjee, Bobby Bhattacharjee, Christopher Kommareddy. Scalable Application Layer Multicast. *Proceedings of ACM Sigcomm 2002, Pittsburgh, Pennsylvania, August 2002.*

[2] Suman Banerjee, Christopher Kommareddy, Koushik Kar, Bobby Bhattacharjee, Samir Khuller. Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications. *IEEE Infocom, April 2003.*

[3] B. Ahlgren, M. Bjorkman, and B. Melande, Network probing using packet trains. *Technical Report, SICS, March 1999.*

[4] M.P. Barcellos, P.D. Ezhilchelvan. An End-to-end Reliable Multicast Protocol Using Polling for Sscalability. *INFOCOM*, 1998.

[5] Supratik Bhattacharyya, Don Towsley and Jim Kurose. The Loss Path Multiplicity Problem for Multicast Congestion Control. *Proceedings of IEEE Infocom, 1999.*

[6] E. Bommaiah, M. Liu, A. McAuley, R. Talpade. AMRoute: Ad-hoc Multicast Routing Protocol. *work in progress, draft-manet-amroute-00.txt,* August, 1998.

[7] M. Castro, M. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman. An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays *IEEE INFOCOM 2003, April 2003.*

[8] Yatin Chawathe, Steven McCanne and Eric A. Brewer. Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. *www.cs.berkeley.edu/ yatin*, 1999.

[9] D.R. Cheriton and S.E. Deering. Host groups: a multicast extension for datagram internetworks. *Proceedings of the ninth symposium on Data communications Pages 172-179,* September, 1985.

[10] Y.H. Chu, S.G. Rao, H. Zhang. A Case of End System Multicast. *ACM Sigmetrics Conference*, pp. 1-12, June, 2000.

[11] Yang Chu , Sanjay Rao , Srinivasan Seshan , Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, August 2001.

[12] S.E. Deering. Multicast routing in internetworks and extended LANs. *ACM SIGCOMM*, pp.55-64, August, 1988.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobsen, C. Liu, L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Network, 4(2)*, April, 1996.

[14] C. Diot, B.N. Levine, B. Lyles, H. Kassan, D. Balsiefien. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, 2000.

[15] H.P. Dommel, J.J. Garcia-Luna-Aceves. Ordered end-to-end multicast for distributed multimedia systems. *Proceedings of the 33rd Annual Hawaii International Conference*, 2000.

[16] Constantinos Dovrolis, Parmesh Ramanathan, David Moore. What do packet dispersion techniques measure? *INFOCOM*, 2001.

[17] A. Fei, J. Cui, M. Gerla, M. Faloutsos. Aggregated Multicast: an Approach to Reduce Multicast State. *Global Internet*, 2001.

[18] S. Floyd, M. Handley, J. Padhye, J. Widmer. Equation-based Congestion Control for unicast applications. *ACM SIGCOMM'2000.*

[19] P. Francis. Yoid Project. *http://www.aciri.org/yoid/*, April, 2000.

[20] R. Gopalakrishnan, Jim Griffioen, Gisli Hjalmtysson, Cormac J. Sreenan, Su Wen. A Simple Loss Differentiation Approach to Layered Multicast. *INFOCOM*, 2000.

[21] D.A. Helder, S. Jamin. Banana Tree Protocol, an End-host Multicast Protocol. *Technical Report CSETR-TR429 -00, University of Michigan*, July, 2000.

[22] S. Jagannathan and K. Almeroth. Using Tree Topology for Multicast Congestion Control. *International Conference on Parallel Processing*, 2001.

[23] S. Jagannathan, K. Almeroth and A. Acharya. Topology Sensitive Congestion Control for Real-Time Multicast. *NOSSDAV*, 2000.

[24] Guillaume Urvoy-Keller and Ernst W. Biersack. A Congestion Control Model for Multicast Overlay Networks and its Performance. *Proc. of NGC, October, 2002.*

[25] K. Lai and M. Baker. Measuring Bandwidth. *INFOCOM*, 1999.

[26] Brian Neil Levine, Jon Crowcroft, Christophe Diot, J. J. Garcia-Luna-Aceves, James F. Kurose. Consideration of Receiver Interest for IP Multicast Delivery. *INFOCOM*, 2000.

[27] S.Lin, D.J.Costello,and M.J. Miller. Automatic Repeat Request Error Control Schemes. *IEEE Communication Magazine,page 5-17*, 1984.

[28] Ketan Mayer-Patel, Lawrence A. Rowe. A Multicast Scheme for Parallel Software-only Video Effects Processing. *ACM Multimedia*, 1999.

[29] Jungle Monkey. *http://www.junglemonkey.net/*

[30] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. *IEEE Globecom 2001, November 2001*.

[31] The Network Simulator - NS2. *http://www.isi.edu/nsnam/ns/*.

[32] K. Obraczka. Multicast Transport Protocols: A Survey and Taxonomy. *IEEE Communications Magazine, 36(1):94-102,*, 1998.

[33] Dimitrios Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An application level multicast infrastructure. *Proceedings of the 3rd USENIX, USITS*, 2001.

[34] L. Rizzo. pgmcc: A TCP-friendly Single-Rate Multicast Congestion Control Scheme. *Proc. of ACM SIGCOMM* , 2000.

[35] S. Shi, M. Waldvogel. A rate-based end-to-end multicast congestion control protocol. *ISCC 2000, Page(s): 678 -686*

[36] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H.Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *In Proceedings of ACM Sigcomm, Aug. 2001*.

[37] Lorenzo Vicisano, Luigi Rizzo, Jon Crowcroft. TCP-like congestion control for layered multicast data transfer. *INFOCOM*, 1998.

[38] Starsky K.Y. Wong, John C.S. Lui. An Architectural Infrastructure and Topological Optimization for End System Multicast. *Technical Report, CS-TR-2001-04.* CUHK, 2001.

[39] J. Yoon, A. Bestavros, I. Matta. Adaptive reliable multicast. *ICC, Vol 3, Pages 1542 -1546*, 2000.

[40] Daniel Zappala. Alternate Path Routing for Multicast. *INFOCOM*, 2000.

[41] Beichuan Zhang, Sugih Jamin, Lixia Zhang. Host Multicast: A Framework for Delivering Multicast To End Users. *INFOCOM*, 2002

[42] Xi Zhang, Kang G. Shin, Debanjan Saha, Dilip D. Kandlur. Scalable Flow Control for Multicast ABR Services. *INFOCOM*, 1999.

[43] Ben Y. Zhao, John Kubiatowicz, Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. *Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley*, April 2001.

[44] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, John D. Kubiatowicz. Bayeux : an architecture for scalable and fault-tolerant wide-area data dissemination. *11th International workshop on on Network and Operating Systems support for digital audio and video*, January 2001.