Adaptive Stream Multicast Based on IP Unicast and Dynamic Commercial Attachment Mechanism: An Active Network Implementation

Seiichiro Tani, Toshiaki Miyazaki, and Noriyuki Takahashi

NTT Network Innovation Labs. Yokosuka, Kanagawa, Japan
{tanizo,miyazaki,nrt}@exa.onlab.ntt.co.jp

Abstract. This paper describes an adaptive IP-unicast-based multicast protocol that dynamically constructs a multicast tree based only on request packets sent by clients. Since the protocol is simple but flexible and does not require any IP multicast addresses or special multicast mechanisms, unlike the IP multicast protocol, it is scalable and suitable for personal stream broadcasting and related services. An application that dynamically attaches advertisements to multicasted streams is also presented. The application attaches advertisements to the streams at active nodes instead of the server so it can deliver advertisements tailored to the individual recipient, according to his/her interests and/or location. An algorithm that minimizes the attachment cost over the corresponding multicast tree is developed. The multicast and advertisement attachment mechanisms are implemented using our own active network environment, and their validity is confirmed.

1 Introduction

The world wide web (WWW) lets users send contents to unspecified recipients and streaming data such as MPEG2 video is now larger percentage of the contents being sent. However, IP-unicast consumes too much bandwidth when delivering streaming data.

IP-multicast is the most popular mechanism for multicasting to recipients located over a wide area. Many multicast routing protocols [6, 7, 10, 11] for IP-multicast have been proposed and are being investigated for stardardization at IETF. However, in order to use IP-multicast, we need special IP addresses, called IP-multicast addresses, to specify the multicast groups. This makes it difficult for individuals to multicast streams since it is necessary to obtain a unique IP-multicast address whenever IP-multicasting is desired. Furthermore, all routers need to be able to route IP-multicast packets.

In the source-specific multicast protocol [3], each multicast group is represented by its server (source) address and a source-specific multicast address, which has to be defined by the standard. Thus, the multicast group can be uniquely specified since the server address is unique. However, all routers still need to recognize IP-multicast addresses.

Recently proposed IP-unicast-based multicast protocols use the active network technology, [13, 12]. The uniqueness of each multicast group is naturally guaranteed since the group is specified by its server address. This solves the problem encountered when individuals start to deliver streaming data. While the branch points of multicast trees need to be active nodes, it is not necessary to modify legacy nodes, i.e,. add special mechanisms, when introducing active nodes to existing networks, since the protocols use only IP-unicast packets. This makes it easy to use the multicast mechanism in a network composed of both active and legacy nodes. The protocol in [12] is so simple that it seems to be scalable. However, it cannot dynamically move branch points when IP-unicast routing paths are changed. The protocol in [13] constructs forward-path-based shortest path trees for high-quality delivery of streaming data. In addition, the protocol can dynamically change branch points by using "ephemeral state probes". However, the server and clients need to exchange many packets to gather information on network topology and to negotiate which branch point should be moved. While some heuristics to reduce such packets were given, the scalability of the protocol would be limited to some extent.

We introduce here a new IP-unicast-based multicast mechanism that dynamically constructs a multicast tree by sharing common links among unicast (reverse) paths from clients to a server. The communication realized by the tree still appears to be unicast for each server-client pair. Thus servers and clients do not have to support a special mechanism for multicast. Our tree construction is based only on a hierarchical keep-alive mechanism, i.e., request packets are periodically sent to the server. The mechanism dynamically reconstructs the tree so that it prevents particular active nodes from being overloaded and can optimize the tree against changes in IP-unicast routing paths. Furthermore, this dynamic reconstruction naturally supports the mobility of servers and clients. The keep-alive mechanism is so simple that it is very scalable. When adopting the hierarchical keep-alive mechanism, one of the most important issues is how to determine time-out. We provide an algorithm that dynamically changes time-out depending on the tree topology in use.

We introduce here an algorithm for attaching advertisements to multicasted streams at active nodes that minimizes the total attachment cost over the corresponding multicast tree. By using the algorithm, each recipient can receive streams with suitable personalized advertisements. From a commercial point of view, this algorithm makes our multicast mechanism more attractive. An analogy is drawn to the advertisement banners on private WWW pages. A recent trend is to customize advertisements to suit each viewer since this amplifies their effectiveness. However, it is difficult to customize advertisements at the server (root) of a multicast tree, since the server has to manage the advertisements sent to all clients. Advertising normally uses a small data set repeatedly. Thus, a reasonable approach is to dynamically load the advertisement data into active nodes and attach the advertisements that meet the preference of each recipient at active nodes. In fact, active networks have been shown to be a desirable approach to adding extra functions to multicast mechanisms [4, 2, 1].

We implemented our multicast mechanism with the advertisement attachment function within our active network environment.

Section 2 describes our multicast mechanism while Section 3 defines two cost models and gives an algorithm for each of them. Section 4 shows our implementation; Section 5 provides a brief conclusion.

2 Multicast Using IP-Unicast Addresses

We present the protocol of our multicast mechanism and then describe its characteristics of load balancing and dynamic reconstruction. Next we consider how to implement the keep-alive mechanism used in the protocol.

2.1 Protocol

Clients who want to receive multicast data send a “join” packet containing the pair of the server address and port as its destination. The legacy nodes on the path between active nodes simply forward multicast packets according to their routing tables since the packets are just IP-unicast packets. When a join packet arrives at an active node on the path between the source and the destination (server), the source address written in the packet is registered with the multicast routing table at the node if the table exists; otherwise the table for the server is created before registration, and the node sends the server the join packet with its address as the source in order to join the multicast tree of the server. This joining operation propagates from the client through intermediate nodes until the join packet reaches the server or a node that already has the table. The tables determine to which client the multicast data from the server is to be delivered. From the above, it is clear that delivery follows the reverse of the unicast path from the client to the server. The reverse-paths between clients and the server are bundled as much as possible by the active nodes on the paths. Clearly, our multicast protocol can work even if the reverse-paths are different from the unicast paths from the server to clients, say, the forward-paths. While, in general, forward-paths give higher stream delivery quality than reverse-paths, forward-path-based tree construction often results in complicated or non-adaptable protocols. Our main goal is a scalable and adaptable protocol, which lets us adopt reverse-path based tree construction.

Another simple but key feature of our multicast tree construction is its keep-alive mechanism. Each client continually sends join packets while it wants to receive multicast data. The parent[1] of the client sends multicast data if a join packet from the client arrives within some interval. In other words, a client that stops sending join packets expires and no multicast data is delivered to the client. The parent node also continues to send join packets to the server of the

[1] Following the usual terminology of a tree, for each node (including the server and clients) of a multicast tree, we refer to the server-side and client-side neighbor(s) as, respectively, the parent and the children of the node. Other related terms like ancestors or descendants will also be used without definitions.

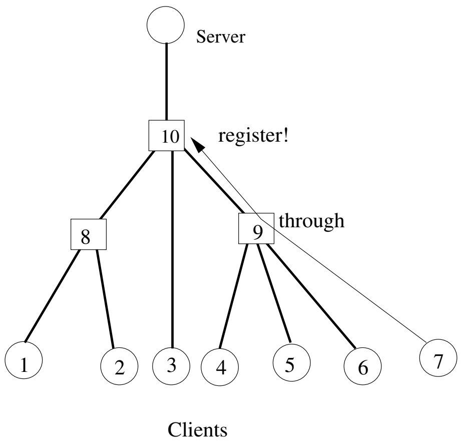


Fig. 1. Client 7 joins when node 9 cannot accommodate any more children.

tree, while it has at least one child that remains active. The same keep-alive mechanism works between the node and its parent. In this way, the keep-alive mechanism starts from clients and passes through at each parent and child pair.

2.2 Load Balancing

Consider the tree in Figure 1. The topmost node is a server and the tree with the server as its root has 3 active nodes (numbered 8 to 10) and 6 clients (numbered 1 to 6). Clients 1 to 6 are already members of the tree. Client 7 now sends a join packet to the server. Here we assume that nodes 9 and 10 can accommodate at most three and four children, respectively. Node 9 has a multicast routing table since it is already a node of the tree. However, the join packet from client 7 is forwarded by node 9 toward the server only if the link to node 10 has sufficient bandwidth, since node 9 has already three children. The join packet arrives at node 10 and client 7 is registered in node 10's table. The resulting tree is shown in Figure 2. This is a natural extension of the basic protocol and it accommodates as many clients as possible without over-burdening active nodes.

2.3 Dynamic Reconstruction of Multicast Trees

In Figure 2, assume that client 5 stops sending join packets. This causes client 5 to expire at node 9. Node 9 can now accommodate one more child. Thus, when the first join packet from client 7 arrives at node 9 after the expiration of client 5, client 7 is registered at node 9's table as shown in Figure 3. Note that client 7 continually sends join packets to stay alive. The join packet from client 7 is no longer forwarded by node 9 toward the server. This leads to the expiration of client 7 at node 10. In this way, the keep-alive mechanism dynamically reconstructs the tree so as to minimize traffic.

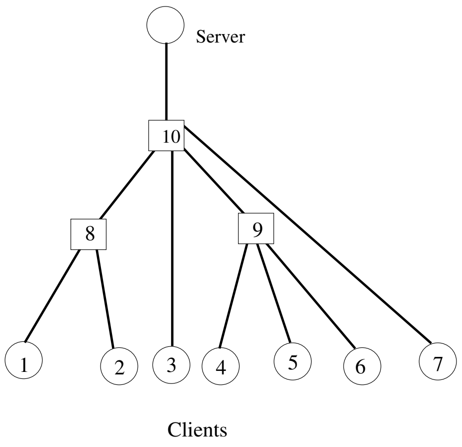


Fig. 2. Tree after client 7 joins.

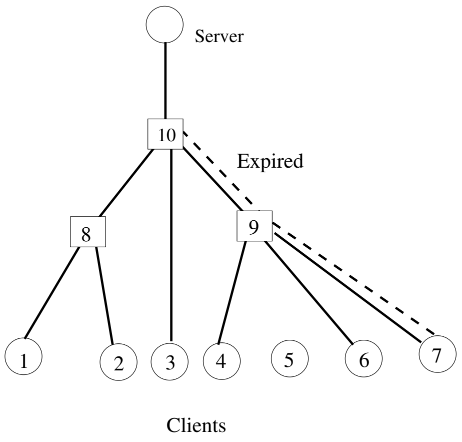


Fig. 3. Tree after client 5 stops sending join packets and expires.

This dynamic reconstruction is effective especially when clients are mobile hosts. In Figure 4, client 3 moves and its neighbor changes from node 10 to node 8. Client 3 is then registered at node 8 and expires at node 10.

Another reconstruction is triggered by the movement of the server as is possible when the server is a mobile host such as a palm-top computer with a small video camera. If the network has a platform to support IP-unicast to mobile hosts, say, the mobile-IP framework [9], the keep-alive mechanism reconstructs the tree as follows. As in Figure 5, the server moves and its neighboring active node changes from node 10 to 11. The join packets from clients are routed for the new location of the server by the mobile-IP framework, since the join packet is

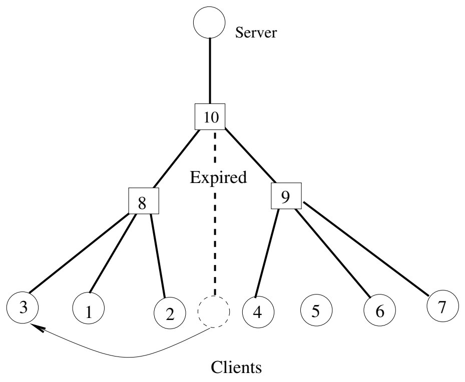


Fig. 4. Tree after client 3 moves.

an IP-unicast packet with the server address as its destination. Here we assume that node 11 comes to lie on the path from nodes 3, 8 and 9 to the server, and that node 10 is not on the path. As a result, nodes 3, 8 and 9 are registered at node 11, and node 11 sends a join packet to the server. Node 10 may also be registered at node 11 since it may send a join packet to the server, when we assume that node 11 is on the path from node 10 to the server. At node 10, nodes 3, 8 and 9 expire after a moment since the join packets from nodes 3, 8 and 9 do not reach 10 any more. Node 10 then stops sending join packets to the server and expires at node 11.

During tree reconstruction, multiple packets carrying the same data may arrive until the old parent-child relationships disappear. In Figure 3, nodes 9 and 10 may send redundant packets to client 6. In Figure 5, nodes 10 and 11 may send the redundant packets to nodes 3, 8 and 9. However, these packets can be discarded by checking the time-stamp and sequence number assigned at the server following, say, RTP (Real-Time Transport Protocol) [8].

2.4 Expiration Time

The keep-alive mechanism must not allow a parent to expire before all of its children. A simple way of accomplishing this is to use a timer at each node: each node periodically sends a join packet to the server if there is at least one child that is active. However, to invocate packet sending by using a timer may so badly load resources that it may delay other processes like data packet delivery.

Another way of implementing the keep-alive mechanism is for each node to send a join packet (if needed)only when the node receives a join packet. In other words, the sending operation is performed as the result of evaluating an active code in a join packet. However, if each active node sends a join packet every time it receives a join packet, the server may receive more join packets than it can process.

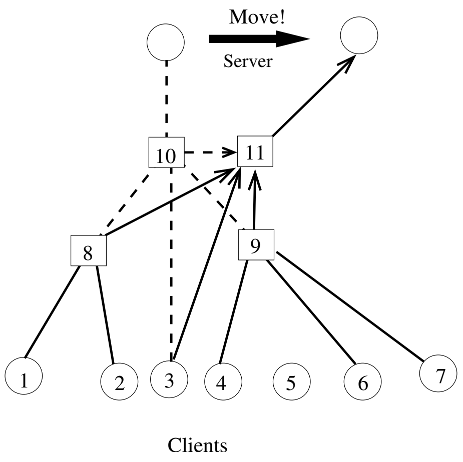


Fig. 5. Tree reconstruction when the server moves.

We now consider when a node should send a join packet and how to set the expiration time interval. Assume that all clients send a join packet periodically at interval D. Each node has a variable T, which is the time when the last join packet was sent. When the node receives a join packet, it compares the current time T_{now} and T. If $(T_{now} - T) \geq D$, it sends a join packet.

Before analyzing the above algorithm, we define the height of a node as the number of links of the longest unicast paths from clients of its descendants to the node. Denote the maximum interval between join packets being sent and received at a node with height h by $D_s(h)$ and $D_r(h)$, respectively.

If we assume that there is no link and processing delay jitter, i.e., $D_r(h) = D_s(h-1)$, it can be proved by induction that

$$D_r(h) \leq h \times D.$$

Assume that a node with height $h-1$ receives a join packet from a child just before time D has passed since the last time a join packet was sent, and the node does not receive join packets until receiving the next join packet from the child. This case maximizes $D_s(h-1)$. Thus, we have $D_r(h) = D_s(h-1) \leq D + D_r(h-1)$. Clearly, $D_r(1) = D$.

If we cannot ignore jitter, we can evaluate $D_r(h)$ by modifying the above slightly. Let ϵ_1 and ϵ_2 be the maximum increase of the link delay and the processing delay, respectively. $D_r(h) \leq D_s(h-1) + \epsilon_1$ and $D_s(h) \leq D + D_r(h) + \epsilon_2$. Thus, we have

$$D_r(h) \leq h \times (D + \epsilon_1 + \epsilon_2)$$

since $D_r(1) \leq D + \epsilon_1 + \epsilon_2$.

In a reliable network in which packets are seldom lost, a straightforward way of setting the expiration time interval is to set $D_r(h^*)$ as the interval at all nodes, where h^* is the height of the server. However, $D_r(h^*)$ is too long for nodes whose height is less than h^*. This may result in consuming bandwidth wastefully, since an active node does not stop sending the streaming data to its children until they expire. A solution to this problem is for each node to dynamically set $D_r(h)$ as the interval for a child with height $h-1$. The node knows the height of its children in a dynamic-programming way by using join packets as follows. Clients write height 0 in the join packets they send. Active nodes choose the maximum value of the heights written in join packets from their children and write the value plus 1 in the join packet they send. Thus, an active node can know the height of its children from the join packets they send. This dynamic way of setting the expiration time interval ensures that the keep-alive-based tree construction remains scalable.

The above procedure can be applied to computing other information such as the number of recipients. The preference vector described in the next section is one such type of information.

3 Attaching Advertisements Customized for Each Client

This section describes an algorithm that attaches advertisements that suit the preference of each client.

Each client (user) chooses its favorite categories from n categories of advertisements (i.e. sports, cooking, travel, etc.) [2]. Without loss of generality, we can assume that all intermediate nodes can be branch points, i.e., active nodes, since we can think of a unicast path between neighboring active nodes as being a link even if there are legacy nodes between the active nodes.

Each active node on the path between the server and each client can attach an advertisement to the stream coming from the parent if no advertisement is attached to the stream; otherwise it can replace/delete the advertisement. This attachment/replacement/deletion operation can be done for each child independent of the other children.

For a network composed of such active nodes, we propose algorithms that attach an advertisement consistent with the preference of each client and minimize the total attachment cost over the tree under the two cost models defined below.

3.1 Binary Cost Model

We describe an algorithm based on the following cost model.

Definition 1 (Binary Cost Model). *It costs an active node 0 to forward the stream coming from its parent to a child, and 1 to attach an advertisement to the stream or replace/delete the advertisement attached by one of its ancestors (and*

[2] Each user does not need to explicitly choose the categories. Such choice can be done automatically by software that extracts the preference of users from their access history.

then send the resulting stream to a child). The active node is charged the cost for each child. In other words, the cost of the active node is the sum of the attachment/replacement/deletion costs for all its children. The cost of a multicast tree is the sum of the costs of all active nodes in the tree.

If an active node with m children forwards the stream to x of the children and attaches/replaces/deletes/ advertisements for $m-x$ remaining children, the cost of the node is $m-x$. Therefore, minimizing the cost of a multicast tree is equivalent to minimizing the number of advertisement attachment/replacement/deletion operations that occur in the multicast tree.

Algorithm. Our algorithm has two phases; request and delivery. In the request phase, information on the clients' favorite advertisement categories (or ad-categories) is propagated to the active nodes on the path to the server. Based on the information, each active node decides which operations should be performed and which ad-category should be used. The above decision is done so that each client can receive streams having the advertisement of its favorite ad-category and the cost of the multicast tree is minimized. If two or more advertisements belong to the same ad-category, one of them is selected in an arbitrary way. In the following, we will use 'advertisement' and 'ad-category' interchangeably. The delivery phase performs the operation decided at each node.

The details of the request phase are as follows.

Each client sends information on its favorite ad-categories expressed by a preference vector (defined below) to its parent. Each active node computes its favorite ad-categories from the preference vectors sent by its children, and sends the preference vector expressing the ad-categories to its parent.

Definition 2 (Preference Vector). *For the number n of ad-categories, a preference vector is an n binary-valued element vector $(f_1, f_2, \ldots, f_n)$ $(f_i \in \{0,1\}, i = 1, 2, \ldots n)$ such that $f_i = 1$ if the ith ad-category is desired, $f_i = 0$ otherwise.*

If no advertisement is attached, we still regard it as one ad-category, say, the first ad-category[3]. Thus, there are two choices at a node, replacement and forwarding.

Each active node computes a new preference vector that will be sent to its parent by (1) summing up the preference vectors of all children, (2) choosing one or more elements having maximum values, and (3) setting 1 on the summation's maximal elements of the new vector and 0 on the other elements. Note that multiple elements may be 1 if there are two or more maximal elements.

For example, consider a node with two children as shown in Figure 6. The two preference vectors from the children are $(0, 1, 0)$ and $(0, 1, 1)$. Preference vector $(0, 1, 0)$ is sent to the parent of the node, since the sum of the two vectors is $(0, 2, 1)$.

[3] Note that the first element is always 0 if clients are prohibited from requesting the first ad-category (this is a necessary condition from a commercial point of view).

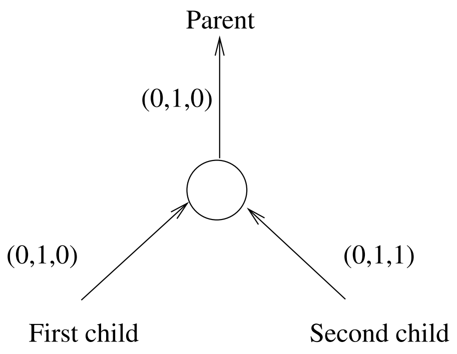


Fig. 6. Preference Vectors into and out of an Active Node.

The node also creates or updates a table used in the delivery phase from the received preference vectors. This table indicates which advertisements the node should replace. This table is called the *delivery table*. For each child sending a preference vector **F**, if the incoming stream has an advertisement for the element with value 1 in **F**, then the forward operation is performed; Otherwise the advertisement is replaced with one of the advertisements for the elements (and the resulting stream is sent to the child).

It is convenient to express the table as an $n \times m$ matrix. Element (i, j) of the table expresses the advertisement with which the advertisement of the incoming streaming data is replaced in order to send the resulting stream to the jth child, when the advertisement of the incoming stream belongs to the ith advertisement. Hence, the table created at the node in Figure 6 is

$$\begin{pmatrix} 2 & 2or3 \\ 2 & 2 \\ 2 & 3 \end{pmatrix},$$

where its children are numbered from left to right[4]. For example, if the streaming data with the first advertisement enters the node in Figure 6, the first row of the above table indicates that the advertisement should be replaced with the second one for sending the resulting data to the first child, and it should be replaced with the second or third one for the second child.

We describe the behavior of the request phase on a multicast tree using Figure 7. Assume that there are three ad-categories. Here, as mentioned before, if no advertisement is attached it is regarded as the first ad-category. In Figure 7, clients 5 and 7 are interested in ad-category 2, and client 6 in ad-category 3. By the definition of preference vectors, a client who is interested in the second ad-category sends preference vector $(0, 1, 0)$ to its parent, and a client who likes the third ad-category sends $(0, 0, 1)$. Active node 3 sends $(0, 1, 0)$ to active node

[4] In the rest of this paper, we adopt the same rule for numbering children.

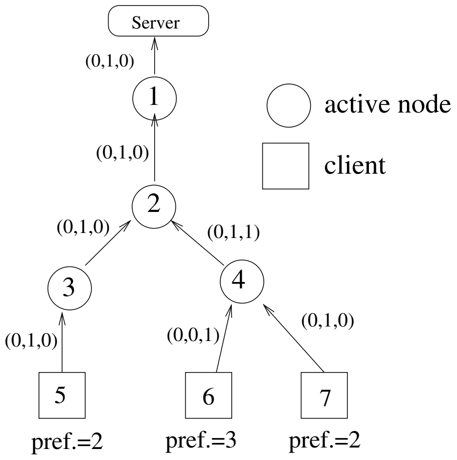


Fig. 7. Request Phase of the Binary Cost Model.

2, active node 4 sends $(0, 1, 1)$ to active node 2, active node 2 sends $(0, 1, 0)$ to active node 1, and active node 1 sends $(0, 1, 0)$ to the server. At this time, the tables created at active nodes 1, 2, 3 and 4 are, respectively,

$$\begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 & 2or3 \\ 2 & 2 \\ 2 & 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 & 2 \\ 3 & 2 \\ 3 & 2 \end{pmatrix}.$$

The delivery phase begins when when preference vectors arrive at the server.

When the server does not have the advertisement function, it regards the preference vectors as just requests for the streaming data it has, and sends the streaming data without any advertisement to its children, i.e., the sources of the requests. When the server can attach advertisements, it sends to each child the streaming data with the advertisement for one of the preference vector elements with value 1.

The streaming data is multicasted with advertisement replacement operations being performed at each node according to the delivery table. Figure 8 shows advertisement replacement. In Figure 8, the number written in each delivery packet expresses the category of the advertisement carried by the packet; the delivery packets are those that carry streaming data with advertisements in the delivery phase. This example assumes that the server does not have the advertisement attachment function, i.e., the server always sends the streaming data with the first ad-category.

The above discussion does not consider the difference in the arrival times of the preference vectors. However, it is unlikely that all preference vectors will arrive at the same time. Another problem is that clients may join or leave the multicast tree at any time. This may change the preference vector to be sent. Yet

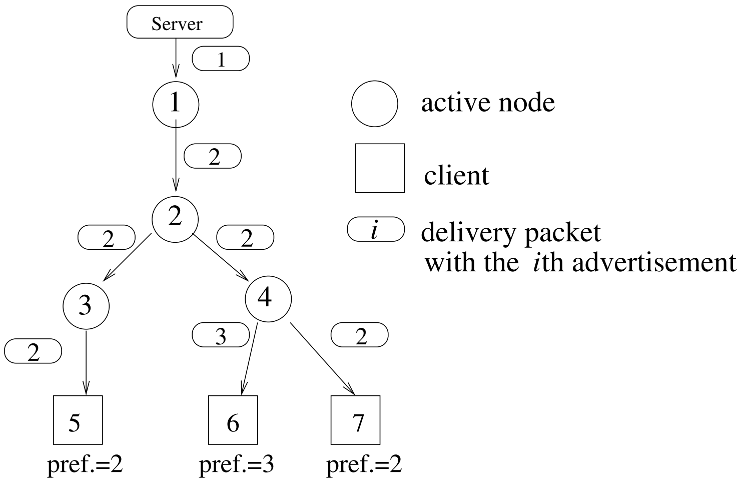


Fig. 8. Delivery Phase of the Binary Cost Model.

another problem is that some clients in the tree may change their favorite ad-categories while receiving the stream. To solve these problems, each node records the preference vector of each child, computes a new preference vector from the recorded vectors, and sends the computed vector to its parent at the time some event occurs, i.e., when a new client joins the tree as a child of the node, when some child leaves the tree, or when the preference vector of a child changes. Note that clients need to send preference vectors when changing their favorite ad-categories as well as when joining the tree. It depends on implementation as to when to check whether the above events have occurred or not. A straightforward way is to check periodically by using timers built in the active nodes, as the case of the keep-alive mechanism in Section 2.4. However, using timers often delays other processes. In our implementation, checking is triggered by the arrival of preference vectors.

In the transitional intervals during which delivery tables are being updated, the advertisement of the incoming streaming data may ‘contradict’ the preference vector that was sent last. However, clients can receive the streaming data with their favorite advertisement provided the table at their parents is updated.

The following theorem guarantees the optimality of our algorithm over stable (i.e. non-transitional) intervals. The theorem can be proved by using inductively the following fact: each active node replaces advertisements in order to minimize the number of advertisement replacements and so minimize the cost of the subtree with the active node as its root.

Theorem 1. *For a given multicast tree, our algorithm minimizes the cost of the tree under the binary cost model over stable intervals.*

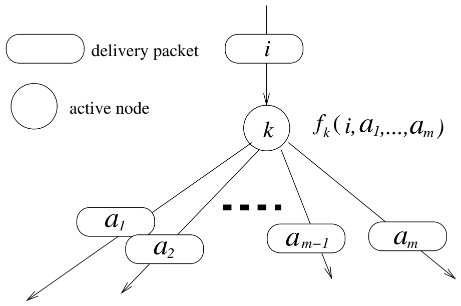


Fig. 9. General Cost Model.

3.2 General Cost Model

In this section, we define the general cost model and generalize an algorithm for minimizing the advertisement attachment cost over a multicast tree under the cost model.

Definition 3 (General Cost Model). *For each active node k, a cost function $g_k(i, a_1, ..., a_m)$ for advertisement replacement exists where m is the number of children of k, i is the category of the advertisement in the incoming streaming data, and a_j $(j = 1, 2, \ldots m)$ is the category of the advertisement with which the advertisement of the incoming data is replaced before being sent to the jth child. The cost of active node k is the value computed by $g_k(i, a_1, ..., a_m)$, and the cost of a multicast tree is the sum of all active nodes in the tree.*

In the above definition, we assume that advertisements of the same category can be regarded as equivalent to each other in terms of the cost function. If not, we divide a category into sub-categories so that we can continue to use the above assumption. We use advertisement and ad-category interchangeably as in the previous section.

Algorithm. Our algorithm has the request and delivery phases. In the request phase, each node (and client) k sends to its parent $P(k)$ a cost vector $\mathbf{V}_k$ defined below and a delivery table is created at each node. The table has the same meaning as in the binary cost model. The delivery phase acts in the same way as in the binary cost model. In the following, we focus on the request phase.

Definition 4 (Cost Vector). *For the number n of ad-categories, a cost vector that node k sends to its parent is an n-dimensional vector $\mathbf{V}_k = (v_1^k, v_2^k, ..., v_n^k)$, where v_i^k is the minimum cost of the subtree $T(k)$ with root k that is attainable when k receives the streaming data with the ith ad-category. The cost vector sent by each client has value 0 at the elements for its favorite ad-categories and infinity, denoted by 'inf', at the other elements.*

For example, a cost vector sent by a client who wants to receive the fourth ad-category is (inf, inf, inf, 0, inf, ..., inf). The first ad-category means that no advertisement is attached as in the previous section. Hence, the vectors from clients have 'inf' at the first element.

Each node k computes v_i^k $(i = 1, 2, \ldots, n)$ from the cost vectors sent by its children and cost function $g_k(i, a_1, \ldots, a_m)$ as follows.

If $c(k, j)$ is the jth child of k, the cost vector from the jth child of k is $\mathbf{V}_{c(k,j)} = (v_1^{c(k,j)}, v_2^{c(k,j)}, \cdots, v_n^{c(k,j)})$. Hence, $v_{a_j}^{c(k,j)}$ is the minimum cost of $T(c(k, j))$ when k sends $c(k, j)$ the streaming data with the a_jth ad-category. Therefore, the minimum cost of $T(k)$ when k receives the streaming data with the ith ad-category and, for each j, sends the jth child the data with the a_jth ad-category, is given by $g_k(i, a_1, \ldots, a_m) + v_{a_1}^{c(k,1)} + \cdots + v_{a_m}^{c(k,m)}$. This leads to:

$$v_i^k = \min_{\substack{a_j \in \{1,\ldots,n\} \\ j=1,2,\ldots m}} \{g_k(i, a_1, \ldots, a_m) + \sum_{j=1}^{m} v_{a_j}^{c(k,j)}\}.$$

By computing this for each i, we obtain $\mathbf{V}_k = (v_1^k, v_2^k, ..., v_n^k)$.

Delivery tables are created/updated during the computation of $\mathbf{V}_k$ such that the ith row of the table is $(a_1, a_2, \ldots, a_m)$ which gives v_i^k for each i $(i = 1, 2, \ldots m)$. By referring to the table, the delivery phase replaces advertisements at each node k so that the cost of $T(k)$ is minimized. Such operation is performed recursively from the root (server) to leaves (clients). Thus we have the following theorem.

Theorem 2. *For a given multicast tree, our algorithm minimizes the cost of the tree under the general cost model over stable intervals.*

If, for an over-loaded node k, we define the cost function of k so that the function gives relatively large values for replacement operations, the forwarding operation is likely at node k. In this way, the load-balancing of active nodes can be accomplished by dynamically updating the cost functions according to the load of node k. The algorithm can also control the traffic through each link. If the link from $P(k)$ to node k is crowded, the algorithm can reduce the traffic of the link by deleting the advertisement at $P(k)$. This is accomplished by updating the cost function of k so that it gives relatively large values when the advertisement in the incoming stream does NOT belong to the first ad-category.

3.3 Applications

It is simple to modify the above algorithms so that information from the server can be used to force the selection of one of multiple advertisements in the same ad-category. Examples of the information from the server include the number of recipients or the kind of movie. By inserting the information into each delivery packet at the server, active nodes can be directed to choose the most appropriate advertisement.

Table 1. An Example of a Vector Table.

child address	preference vector	arrival time
10.27.124.5	01010010	20:19:18
10.27.124.7	01001000	20:19:15
10.27.124.1	00100000	20:19:20

Table 2. Functions and Source/Destination Addresses of Request/Deliver Packets.

Packet name	Source addr.	Destination addr.	Function
Request packet	Client *or* Active node	Server	Join the multicast tree, Keep alive, *and* Carry a preference vector
Delivery packet	Server	Client *or* Active node	Carry the streaming data

In addition, the proposed algorithms have other applications. Transcoding (transforming encoding formats) at active nodes is one such application. The preference vector can be utilized to carry the clients' desired format back up the tree. The same mechanism used to minimize the cost of replacement operations will also minimize the cost of transcoding operations.

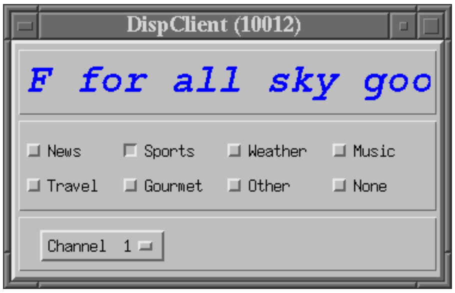
4 Implementation

To implement our algorithm, we used our active network environment which is based on a 100BASE-T Ethernet. The architecture of the environment is based on the "active packets approach" [5]; the active code contained in a active packet can read from and write to the memories of each node. Memory contents are kept after the evaluation of a packet is completed.

In our commercial multicasting implementation, a server sends out an MPEG2 data stream and clients receive the stream with their favorite advertisements. We used characters as the advertisement data in the first trial, and the implemented advertisement attaching algorithm is based on the binary cost model. We used two kinds of active packets: request packets and delivery packets, as shown in Table 2.

The active packet in the request phase includes a preference vector, and the delivery packet includes the MPEG2 streaming data and the advertisement data. We call the packets carrying a preference vector the *request packets*. They also have the join-packet function for multicast-tree construction. In other words, request packets are sent at interval D by the clients and are used to dynamically construct multicast trees (i.e., create/update the multicast routing table) as shown in Section 2, as well as to create/update the delivery tables. Thus

Viewer.



Control Panel.

Fig. 10. A Snapshot of the Client Interface.

the source and destination of the request packet is the client and the server, respectively.

When request packets arrive at an active node, the preference vectors contained in the packets and their arrival time are recorded for each client in a table, called a *vector table*. Table 1 is an example of a vector table. The delivery table then is created or updated. Next, a new preference vector is computed from the preference vectors recorded in the vector table according to the method in section 3.1. Finally the new vector is sent to the server (by using a request packet) if it is different from the last vector sent or time D has passed since the last time a request packet was sent. At this time, the source and destination of the request packet is the active node and the server, respectively. These above operations are performed by evaluating the active code in each request packet.

Delivery packets are initially sent by the server to its children (i.e. active nodes or clients). When a delivery packet that has the ith ad-category arrives, the advertisement replacement operation is performed (if needed) for each live child j according to the (i, j) element of the delivery table, and the delivery packet is sent to the jth child, where the source and destination of the delivery packets is the server and the jth child, respectively. Whether a child is alive or not is determined by comparing the current time with the arrival time of the last request packet from the child. These actions are performed by evaluating the active code in the delivery packet.

Note that the vector table and the delivery table as well as the multicast routing table are required for each multicast tree (i.e. each server). In our implementation, these tables are merged into one table for space efficiency.

Figure 10 shows the user interface of the client program. The right panel (control panel) is for selecting favorite ad-categories and choosing channels. Favorite ad-categories can be changed while receiving the streaming data. The advertisement is displayed on the upper part of the control panel. The left panel

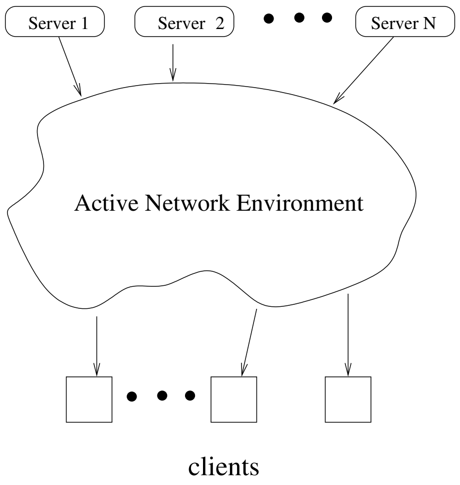


Fig. 11. Experimental Environment.

is a viewer that displays the streaming data (MPEG2 movies). Each channel corresponds to a multicast tree. In our experiment, we prepared several servers as shown in Figure 11. Any of the streaming data from these servers can be displayed in the client program by selecting the corresponding channel. Feasibility testing confirmed that implementing the algorithms did not create a bottleneck in terms of CPU performance.

5 Conclusion

With IP-unicast-based multicasting, individuals can easily multicast streams, since the global uniqueness of each multicast group is naturally guaranteed by server address and port. Furthermore, it is not necessary to modify legacy nodes when introducing active nodes as branch points to existing networks in order to service IP-unicast-based multicasting.

In this paper, we proposed a scalable and adaptable IP-unicast-based multicast protocol. The basic idea of our protocol is to dynamically construct a multicast tree by sharing common links among unicast reverse-paths between a server and clients. The multicast communication realized by this mechanism still appears to be unicast for each server-client pair. This aspect releases servers and clients from needing any special multicast mechanism. The keep-alive technique, a key feature of our protocol, enables the load-balancing of active nodes and dynamic tree reconstruction. It uses only keep-alive packets, i.e., request packets periodically sent to the server. This simple mechanism makes our protocol scalable. The dynamic tree reconstruction so realized naturally supports the mobility of servers as well as clients. In order to reduce the load of the nodes, the keep-alive mechanism does not use timers, only keep-alive packets need be sent to active nodes. This timerless approach is realized by an algorithm that dynamically sets the appropriate time-out at each active node.

To make our multicast more attractive, we developed an algorithm that attaches at active nodes advertisements that fit the preference of each recipient. The algorithm determines which active nodes should attach which advertisements in order to minimize the total attachment cost over each multicast tree. The advertisement attachment algorithm is so general and effective that it can be applied to other services such as transforming encoding formats (transcoding).

We implemented our multicast mechanism with the advertisement attachment function within our active network environment. Our work so far has demonstrated the validity of the idea. In the future, we plan to quantitatively evaluate the mechanisms' performance.

References

[1] H. Akamine, N. Wakamiya, M. Murata, and H. Miyahara. An approach for heterogeneous video multicast using active networking. In *Proceedings of IWAN 2000*. IFIP, 2000.

[2] B. Duysburgh, T. Lambrecht, B. Dhoedt, and P. Demeester. Date transcoding in multicast sessions in active networks. In *Proceedings of IWAN 2000*. IFIP, 2000.

[3] H.W. Holbrook and D.R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source application. In *Proceedings of SIGCOMM*, 1999.

[4] L.H. Lehman, S.J. Garland, and D.L. Tennenhouse. Active reliable multicast. In *Proceedings of INFOCOM'98*. IEEE, 1998.

[5] K. Psounis. Active networks: Applications, secuirity, safety, and architectures. *IEEE Communicatons Surveys*, pages pp. 2–16, 1999.

[6] RFC1075. Distance Vector Multicast Routing Protocol. IETF Home Page: `http://www.ietf.org`.

[7] RFC1584. Multicast Extensions of OSPF. IETF Home Page: `http://www.ietf.org`.

[8] RFC1889. RTP: A Transport Protocol for Real-Time Applications. IETF Home Page: `http://www.ietf.org`.

[9] RFC2002. IP Mobility Support. IETF Home Page: `http://www.ietf.org`.

[10] RFC2117. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. IETF Home Page: `http://www.ietf.org`.

[11] RFC2189. Core Based Trees (CBT version 2) Multicast Routing. IETF Home Page: `http://www.ietf.org`.

[12] I. Stoica, T.S. Eugene, and H. Zhang. Reunite: A recursive unicast approach to multicast. In *Proceedings of the INFOCOM 2000*. IEEE, 2000.

[13] S. Wen, J. Griffioen, and K.L. Calvert. Building multicast services from unicast forwarding and ephemeral state. In *Proceedings of OPENARCH 2001*. IEEE, 2001.