Abstract—In this paper, we propose a bandwidth-efficient multicast routing mechanism using rectilinear Steiner trees for Internet Protocol (IP)-based low earth orbit (LEO) satellite networks. Different from existing work that minimizes the end-to-end delay, our mechanism minimizes the total bandwidth, i.e., the number of hops, for a multicast tree since the wireless bandwidth in satellite networks is a limited and scarce resource. We formulate the problem as an optimization problem and propose a distributed algorithm and a protocol to support the dynamic group membership for multicasting over LEO satellite networks. Our simulation results show that the trees created by our mechanism consume less bandwidth than the shortest path trees. The difference between the optimal solutions and the solutions obtained by our distributed algorithm is within 5%. Furthermore, our protocol has a limited overhead and can support a large number of group members.

Index Terms—Low earth orbit (LEO) satellite, multicast, rectilinear Steiner tree (RST).

I. INTRODUCTION

The low earth orbit (LEO) satellite network is an excellent solution to providing broadband Internet services to globally scattered users due to its global coverage, broadcasting capability, and bandwidth-on-demand flexibility. Such a network consists of satellites with onboard processing and intersatellite links (ISLs). Since satellites travel at high speed, the network needs an efficient routing algorithm to handle the dynamic changes in the network topology.

Previous routing approaches for asynchronous transfer mode (ATM)-type onboard satellites [1], [2] are based on connection handover mechanisms to tackle the topology change. As the Internet has become popular, the concept of logical locations is introduced [3] to provide connectionless service over Internet Protocol (IP)-based LEO satellite networks. The idea is that the surface of the Earth is divided into multiple logical locations, and each location represents a node in the network. Each logical location is associated with the nearest satellite at the time. Each time a satellite leaves the region, it delivers the whole routing table to the successor satellite. Each intraplane ISL vertically connects two neighbor logical locations in the same orbit, and each interplane ISL horizontally connects two neighbor logical locations in the neighbor orbits. Packets move toward the destination logical location, regardless of which satellite is associated with the location. Therefore, this approach eliminates the overhead of connection handover and is suitable for connectionless IP networks.

Satellite networks are suitable to provide multicast services. Previous multicast routing protocol for satellite networks [4] aims to provide multicast service for real-time applications over LEO satellite networks. It uses the minimum-hop path between the root and each member to minimize the end-to-end delay for real-time multimedia services. So far, very little work has been designed for nonreal-time services. The multicast trees for nonreal-time services should be designed to minimize the amount of used bandwidth, instead of the end-to-end delay, since the wireless bandwidth of LEO satellite networks is a limited and scarce resource.

In this paper, we adopt the concept of logical locations and focus on nonreal-time applications. We propose a bandwidth-efficient multicast routing mechanism using rectilinear Steiner trees (RSTs) for IP-based LEO satellite networks. An RST is a connected acyclic graph with a minimum total length. Each edge of the tree is either a vertical or a horizontal line segment, which contains a single or multiple ISLs. Finding an RST in a graph is NP-complete [5]. In [6], the authors prove that an RST can be approximated arbitrarily close to the optimal solution. In [7], an approximation algorithm is proposed for a bounded-slew RST, where the difference in the lengths of two paths connecting the root and any two leaves is bounded. The authors in [8] minimize the kth moment of an RST, where the length is not linearly summed up in this case. In [9], the authors present a class of probability-based approaches toward the best solutions to RST under statistical sense.

In this paper, we formulate the problem of finding RSTs in LEO satellite networks. As we know, there is no specific formulation for RSTs in LEO satellite networks. We then propose a distributed algorithm. Previous algorithms [6]–[10] are centralized algorithms that are designed for planar graphs. Therefore, they can be used in Manhattan street networks, but they are not suited for LEO satellite networks since such networks have the seam and the polar regions. Moreover, previous algorithms require centralized computation, but our algorithm is distributed. Based on the distributed algorithm, we design a multicast routing protocol that supports the dynamic group membership in LEO satellite networks. Although we minimize the bandwidth consumption of a multicast tree in this paper, we can also consider load balancing in the network by including the load of each link in the calculation of the cost of each link.
[11], and the problem becomes to minimize the total cost of the tree. Without loss of generality, we focus on the minimization of bandwidth consumption in this paper and assign unit cost to each link.

The rest of this paper is organized as follows: The problem is formulated in Section II, and the distributed algorithm is described in Section III. The protocol based on the distributed algorithm is presented in Section IV, and the simulation results are shown in Section V. Finally, this paper is concluded in Section VI.

II. ILP FORMULATION

In this section, we model the LEO satellite network as a connected directed graph. Each node denotes a logical location in an IP-based LEO satellite, and each arc is either an intraplane or an interplane ISL. A multicast tree can be a source-based tree or a shared tree. A Steiner node is an on-tree node with more than two incident arcs. A corner node is an on-tree node with two noncollinear arcs. A group member is a sender or a receiver in the multicast tree and can be a Steiner or a corner node. For example, the multicast tree in Fig. 1 is an RST. A segment $s_{u,v}$ is a horizontal or a vertical on-tree line from node $u$ to $v$, where each of the two endpoints $u$ and $v$ must be a group member, a Steiner node, or a corner node. A segment may cross the seam or the polar region. To handle the polar regions in this paper, we enable a vertical segment to extend from a group member and pass through the north or south polar region to any other node in the same plane. For example, $s_{c,h}$ is a segment, but $s_{c,h}$ is not a segment. A segment consists of a set of arcs in the segment. For example, arc $a_{w,u}$ belongs to segment $s_{a,y}$. A complete segment is a segment that cannot be extended. For example, segment $s_{a,y}$ is a complete segment, but $s_{w,y}$ is not. An extending segment of a group member is a segment, and an endpoint of which must be the group member, and only the two endpoints of the segment can be the group members. For example, segments $s_{a,c}$, $s_{m,w}$, and $s_{u,y}$ are extending segments of nodes $a$, $m$, and $y$. Segment $s_{c,w}$ is not an extending segment even if node $c$ is a group member, because member $m$ is not an endpoint of the segment. An extending tree is a multicast tree formed by a set of extending segments. The multicast tree in

Fig. 1 is an extending tree. An extending RST is an extending tree with a total length equal to any RST spanning the same group members.

In this section, we propose a new integer linear programming (ILP) formulation for the RST problem in LEO satellite networks based on the following lemma [12].

**Lemma 2.1:** In Manhattan street networks, for any set of group members, there exists an RST that can be decomposed into multiple subtrees joined only at members, where each subtree is in one of the forms shown in Fig. 2.

The first subtree in Fig. 2(a) includes a corner node connecting to two group members with two perpendicular segments. In only one of the two segments, we have other group members connecting to the segment from each side. The second subtree in Fig. 2(b) contains a segment connecting two members, whereas other members connect to the segment from each side. The subtree in Fig. 2(c) is similar to the subtree in Fig. 2(a), but the other of the two segments now has an additional group member connecting to the segment.

**Lemma 2.2:** In Manhattan street networks, there exists an extending RST for any set of group members.

**Proof:** Each subtree in Fig. 2 is an extending tree. According to Lemma 2.1, there exists an RST that can be decomposed into multiple subtrees joined only at members. Therefore, there exists an extending RST for any set of group members. \[\square\]

**Theorem 2.3:** In LEO satellite networks, there exists an extending RST for any set of group members.

**Proof:** We prove the theorem by designing an algorithm to reroute any RST to a new RST. We prove that the new RST is an extending RST because there is at least one group member in each complete segment. For each complete segment in any RST, if there is no group member in the segment, the number of incident segments extending to or from each orthogonal direction must be identical; otherwise, the segment can be slid in the direction with more orthogonal segments to reduce the total length of the tree. Therefore, we can slide the segment to either orthogonal direction without increasing the total length of the tree, by which the segment must hit a group member.

1 A segment is slid in an orthogonal direction if the segment moves toward the direction that is orthogonal to the segment. Each segment incident on the slid segment is either shortened or lengthened, and some segments may be created during the slid operation. For example, in Fig. 1, segment $s_{m,w}$ is lengthened when segment $s_{u,y}$ is slid upward to $s_{y,t}$, and segment $s_{m,v}$ is created when segment $s_{c,w}$ is slid to the right to $s_{d,w}$.
a corner node, or a Steiner node. In the latter two cases, the segment can be extended and becomes a new longer complete segment; otherwise, the total length of the tree can be reduced.

If there is no group member in the new complete segment, the aforementioned statement must also hold. Since the length of a complete segment is limited, there is at least one group member in each complete segment in the final RST. Therefore, the final RST is an extending RST.

Our formulation is to find an extending RST for a given multicast group in a LEO satellite network. For each group member, our formulation determines the path from the root to the member. Each path must only contain the extending segments of group members. The extending RST is the union of the paths from the root to all group members such that the total length is minimized. Since a member can be either a sender or a receiver, packets can be sent in either direction of each arc in the tree.

The input parameters of our formulation are given as follows:

- \( V \) set of nodes in the LEO satellite network;
- \( a_x, a_y \) plane ID and the satellite ID of node \( u, u \in V \);
- \( A \) set of arcs in the LEO satellite network;
- \( A_t \) set of arcs in multicast tree \( t \);
- \( a_{u,v} \) arc from node \( u \) to \( v \), \( a_{u,v} \in A \);
- \( n_X \) number of planes in the LEO satellite network;
- \( n_Y \) number of satellites in each plane;
- \( r_t \) root of multicast tree \( t \);
- \( M_t \) set of members of multicast tree \( t \);
- \( s_{u,v} \) segment from node \( u \) to \( v \), and \( s_{u,v} \) can pass through the polar region if \( u \) and \( v \) are located in the same plane;
- \( S^t_m \) set of extending segments of \( m \) in \( t \), \( m \in M_t \);
- \( C_t \) set of corner nodes in \( t \);
- \( S_t \) set of extending segments of all members in \( t \), namely \( S_t = \bigcup_{m \in M_t} S^t_m \).

The variables of our formulation are given as follows:

- \( \phi^t_{m,u,v} \) binary variable, \( m \in M_t, s_{u,v} \in S_t \); if \( \phi^t_{m,u,v} \) is one, segment \( s_{u,v} \) is on the path from \( r_t \) to \( m \), and the path is used by the extending tree; otherwise, \( \phi^t_{m,u,v} \) is zero;
- \( \tau^t_{u,v} \) binary variable, \( m \in M_t, a_{u,v} \in A, a_{u,v} \) must belong to an extending segment of at least one member; if \( \tau^t_{u,v} \) is one, arc \( a_{u,v} \) is on the extending tree; otherwise, \( \tau^t_{u,v} \) is zero.

Our formulation has the objective function \( \min_{a_{u,v} \in A} \sum \tau^t_{u,v} \), which minimizes the total length of the extending tree. Our formulation is subject to the following constraints:

\[
\sum_{v : s_{u,v} \in S_t} \phi^t_{m,u,v} - \sum_{v : s_{u,v} \in S_t} \phi^t_{m,v,u} = 1 \\
\forall m \in M_t, \ u = r_t, \ m \neq r_t \tag{1}
\]

\[
\sum_{u : a_{u,m} \in S_t} \phi^t_{m,u,m} - \sum_{v : a_{u,v} \in S_t} \phi^t_{m,m,v} = 1 \\
\forall m \in M_t, \ m \neq r_t \tag{2}
\]

\[
\sum_{v : s_{u,v} \in S_t} \phi^t_{m,u,v} - \sum_{v : s_{u,v} \in S_t} \phi^t_{m,v,u} = 0 \\
\forall m \in M_t, \ \forall u \in V - M_t - \{r_t\} \tag{3}
\]

The first three constraints are the flow-continuity constraints. Constraints (1)–(4) guarantee that there is a path from the root to each group member, and the path is in the extending tree. For each pair of arcs spanning the same nodes but in opposite directions, constraint (5) guarantees that at most one of the two arcs is in the extending tree. Constraint (5) also enforces that at most one path from the root to each group member can be used by the extending tree. In Fig. 1, assuming that node \( o \) is the root of \( t \), variables \( \phi^t_{m,a,c} \) and \( \phi^t_{m,c,m} \) are one, and the variables corresponding to other segments are zero for group member \( m \). Therefore, \( \tau^t_{a,b}, \tau^t_{b,c}, \tau^t_{c,h} \), and \( \tau^t_{h,m} \) are all one. For nodes \( w \) and \( x \) and group member \( y \), constraint (5) enforces that at most one of \( \phi^t_{y,w,y}, \phi^t_{y,u,y}, \phi^t_{y,y,u}, \phi^t_{y,y,u} \), and \( \phi^t_{y,y,u} \) can be one in any feasible solution.

In real LEO satellite networks, the links over the seam and in the polar region tend to suffer more packet losses since adjacent satellites move at relatively higher speeds in these regions. Therefore, to address the impact of satellite mobility, a multicast tree should avoid using too many links over the seam or in the polar region by assigning a higher cost to each of such links. For example, we can substitute each real link over the seam or in the polar region with multiple concatenated virtual links to give a larger cost to each real link. Our ILP formulation and algorithm will adopt fewer real links over the seam and in the polar region to reduce the objective value of the problem.

III. DISTRIBUTED ALGORITHM

Although our formulation can find the optimal solutions, it requires centralized computation and is not suitable for dynamic traffic. In this section, we design a distributed algorithm that supports dynamic group membership. In our algorithm, a new member first finds the shortest path to the closest group member that has joined the tree. The new member then connects to the closest on-tree node in the shortest path. Here, we assume that the root maintains a record of group membership for admission control, as in [4]. Therefore, the root can find the group member closest to the newly joining member. Since the routing of a multicast tree may deteriorate after a series of joins or departures, our algorithm reroutes the multicast tree to reduce the total length of the tree.

In our algorithm, we first identify a rectangular region to reroute the tree in the region. Then, we try to slide a segment located on the boundary of the region toward the other side of
the region. Other segments incident to the slid segment need to be either extended or shrunk, and we can reduce the total length of the tree when the number of extended segments is smaller than the number of shrunk segments. In addition, the total length of the tree further decreases after the slid segment is merged with the one on the boundary of the other side of the region. We describe the details of our algorithm in the rest of this section.

A rerouting procedure is initiated in a rerouting region. A rerouting region \( R_{a,b,c,d} \) is a subgraph containing the nodes and arcs within or on the boundary of a rectangular region whose four corners are \( a, b, c, \) and \( d \). In each rerouting region, 1) at least three corners of the rectangle must be Steiner or corner nodes, 2) at most one complete segment can be on each boundary, and 3) at least one corner of the rectangle must be in the complete segment. Rerouting region \( R_{a,b,c,d} \) may contain part of the seam or the polar region, and note that a rerouting region crossing the polar region can be transferred into the corresponding region in our algorithm. Before a rerouting procedure is initiated, only the nodes and arcs on the boundary of the rerouting region can be in the tree, and the subtree in the rerouting region must be connected. In Fig. 3(a), for example, \( R_{a,g,i,c} \) is a rerouting region, but \( R_{b,h,i,c} \) is not a rerouting region since node \( b \) is not connected to node \( c \) with the on-tree arcs in the region. Region \( R_{g,j,l,i} \) is not a rerouting region because on-tree arc \( a_h,k \) is within the region. In Fig. 3(d), \( R_{h,i,f} \) is not a rerouting region because only two corners of the rectangle are Steiner or corner nodes on the tree.

Our distributed algorithm incrementally reroutes the multicast tree. Since it requires no on-tree arc within the rerouting region before a rerouting procedure begins, our algorithm first reroutes the subtrees in smaller rerouting regions to reduce the number of corner nodes and zigzag segments such that we can later reroute the subtrees in larger rerouting regions. Our distributed algorithm reroutes the multicast tree only in rerouting regions to reduce the information stored at each on-tree node when the algorithm is implemented as a protocol. Each Steiner or corner node stores or queries the nodes on the boundaries of at most four rerouting regions located on the four quadrants of the node. Moreover, each Steiner or corner node can independently and asynchronously initiate a rerouting procedure. Although our distributed algorithm locally reroutes the multicast tree, our simulation results show that the difference in the total length between the RSTs and that generated by our algorithm is within 5% on average.

After identifying a rerouting region, our algorithm slides an on-tree segment on a boundary toward the opposite boundary if the total length of the new tree can be reduced. When a segment is slid, other segments incident to the slid segment may need to be extended or shrunk to maintain connectivity of the multicast tree. We can reduce the total length of the tree when the number of extended segments is smaller than the number of shrunk segments. In addition, the total length of the tree further decreases after we slide the segment to the other sides because we merge the segment with the one on the boundary of the other side of the region. Fig. 3 gives an example for the rerouting procedure. Fig. 3(a) shows the initial multicast tree. In Fig. 3(b), \( s_{g,i} \) in \( R_{a,g,i,c} \) is slid to the opposite boundary. One segment \( s_{h,k} \) is extended to node \( b \) to maintain the connectivity of the tree, whereas two segments \( s_{d,g} \) and \( s_{i,c} \) are shrunk. In addition, the slid segment \( s_{g,i} \) is merged with \( s_{a,b} \) to further reduce the total length of the tree. In Fig. 3(c), our algorithm slides \( s_{a,d} \) in \( R_{a,d,c,b} \) to the opposite boundary. Fig. 3(d) gives the final multicast tree, which is also an optimal solution. In Fig. 3(a), we can also slide \( s_{a,g} \) in \( R_{a,g,h,b} \) to the opposite boundary first and then slide \( s_{h,i} \) in \( R_{h,b,i,c} \) to the opposite boundary.

Fig. 4 gives the details of our rerouting procedure that slides a segment \( s_{b,c} \) in \( R_{a,b,c,d} \) to the opposite boundary \( s_{a,d} \). Step 1 adds the other three boundaries to the tree to maintain...
Given: $V, A, M_t$, and $r_t$. 

Find: routing of multicast tree $t$.

1. Let $M'_t \leftarrow \{r_t\}$;
   
   for each member $m \in M'_t$, 
   
   let $n, p \leftarrow \arg \min_{n,p} \left\{ |s_{n,p}| + |s_{p,m}| \mid n \in M'_t, p \in V \right\}$;

   let $P$ denote the path from the closest on-tree node to $m$, $P \subseteq s_{n,p} \cup s_{p,m}$, let $A_t \leftarrow A_t \cup \{P\}$;

   let $M'_t \leftarrow M'_t \cup \{m\}$;
   
   end for.

2. For each node $m \in S_0 \cup C_t$,
   
   for each $n$ and $p$ such that $n, p \in M_t \cup S_t \cup C_t$,

   $s_{m,n} \cap s_{m,p} \subseteq A_t$, and $R_{p,m,n,q}$ can be a rerouting region with $s_{m,n}$ and $s_{m,p}$ as the two orthogonal boundaries,

   Sliding $(R_{p,m,n,q}, s_{m,n}, A_t)$;

   end for;

   end for;

   repeat the step until no node can reduce the total length of $t$.

Fig. 5. Distributed algorithm.

connectivity of the tree. Segment $s_{a,d}$ must be located in the rerouted tree because $s_{b,c}$ is slid to $s_{a,d}$. However, part of segments $s_{a,b}$ and $c_{e,d}$ may disappear because the two segments are shrunk when we slide $s_{b,c}$, and we identify the disappearing part of the two segments at Steps 2 and 3. Specifically, for each of the two shrunk segments, we find the first reached Steiner node or group member $m$ when we slide $s_{b,c}$, and $m$ is the terminal of the disappearing segment. Step 4 extends the segments incident to the slid segment to the opposite boundary to maintain the connectivity of the tree. The final step substitutes the original tree with the rerouted tree if the total length of the tree can be reduced. In this section, we assume that each arc and segment are undirected.

Fig. 5 summarizes our distributed algorithm. Each member at Step 1 sequentially joins the multicast tree. Each member first finds the shortest path to the closest group member that has joined the tree. The new member then connects to the closest on-tree node with the shortest path. After all members have connected to the multicast tree, each Steiner and corner nodes at Step 2 independently use the rerouting procedure to reduce the total length of the multicast tree. The node uses two orthogonal incident segments to specify a rerouting region. The algorithm stops after no node can reduce the total length of the multicast tree by the rerouting procedure.

Our algorithm is loop free since the rerouting procedure is loop free. Before a rerouting procedure starts, any node in a rerouting region must be connected to other nodes in the rerouting region with only the segments on the boundary. When we slide a boundary segment to the opposite side of the rerouting region, all incident segments from or to the outside the region are extended, and other incident segments are removed or shrunk. Therefore, after the rerouting procedure, there is only one path connecting any two nodes in the rerouting region, and all segments in the path are in the rerouting region. Therefore, the multicast tree is loop free after a rerouting procedure.

The time complexity is $O(|V||A|)$, which counts all sliding operations and not only those that are successful. Our algorithm enforces that only the nodes and arcs on the boundary of the rerouting region can be located in the tree. In other words, any two rerouting regions cannot overlap because a rerouting region cannot include on-tree nodes or arcs within the region. Therefore, each arc can be located on the boundaries of at most two rerouting regions at any time. Since each rerouting region incurs a success or a failure rerouting procedure, each arc participates in at most two rerouting procedures at each step, whereas the running time shared by each arc in a rerouting procedure is $O(1)$. Therefore, we need $O(|A|)$ time to complete the rerouting procedures of all rerouting regions, and the rerouting procedure in at least one rerouting region must succeed at each step. We need $O(|V|)$ steps because the total length of the tree is at most $|V|$. Therefore, the running time of our algorithm is $O(|V||A|)$.

The rerouting procedure can be implemented in a distributed manner because each Steiner and corner nodes independently reduce the total length of the multicast tree. Moreover, the procedure requires the Steiner and corner nodes to remember or query only the topology of the local multicast tree instead of the whole tree. Therefore, our algorithm can be implemented as a multicast routing protocol.

IV. PROTOCOL DESIGN

In this section, we propose a protocol based on the distributed algorithm described in Section III. To iteratively reroute the multicast tree in the distributed manner, we regard each corner node, Steiner node, and member as a rerouting node of the tree, where each rerouting node stores the information about the subtree in the rerouting region of each quadrant. Within the rerouting region, each rerouting node can perform the rerouting procedure of our distributed algorithm to reduce the bandwidth consumption of the multicast tree. In the following, we first explain the state stored in each rerouting node and define the control messages of our protocol. We then describe the procedures of joining/leaving a multicast group, obtaining the routing of a subtree and sliding a segment in a rerouting region.

Each node in the multicast tree stores the identities of the parent node and the child nodes in the tree. In addition, each node that is also a rerouting node stores the subtree in the rerouting region of each quadrant. Each subtree is specified by the rerouting nodes in the tree and segments connecting the rerouting nodes. Each rerouting node in the subtree is associated with a subtree size to decide the rerouting initiator of the corresponding rerouting region. The rerouting initiator decides whether the subtree can be rerouted with the rerouting procedure in Section III. The rerouting initiator needs some rerouting nodes to reroute the subtree if the size of the subtree can be reduced. We later explain the procedures for the rerouting initiator and for subtree rerouting in this section.

Our protocol defines the following control messages.

1) Join: A Join message contains the set of segments that the message must traverse. A new member sends a Join message to join the multicast tree. A rerouting node sends a Join message to create new segments on a subtree during a rerouting procedure.
2) Grant: An on-tree node sends a Grant message in reply to the node that sends a Join message.
3) Leave: An on-tree node sends a Leave message to its parent node if it has no child node in the tree and no member host in its coverage area. A rerouting node sends a Leave message to remove existing segments in a subtree during a rerouting procedure.
4) Invite: The rerouting initiator sends an Invite message to request a rerouting node to create new segments during a rerouting procedure. The created segments are specified in each Invite message such that the receiver of the message can send a Join message to create the segments.
5) Update: Each rerouting node obtains the routing of the subtree in a rerouting region from the Update messages disseminated by the rerouting nodes of the subtree.

A. Joining/Departing a Multicast Tree

When a new member decides to join a multicast tree, it first obtains the identity of the closest on-tree member toward the root of the tree. The member then sends a Join message in which the destination is the identified on-tree member. The message stops when it reaches the identified on-tree member or any intermediate on-tree node. The on-tree node then returns a Grant message to the new member, and any node traversed by the message creates a state for the multicast tree.

When a member decides to leave the group, it sends a Leave message to the parent node if the member has no child node in the tree. Each node that receives a Leave message also leaves the tree and passes the message upstream if it is not a member in the tree and has no other child node.

B. Obtaining the Routing of a Subtree

Each rerouting node obtains the routing of the subtree in a rerouting region from the Update messages disseminated by the rerouting nodes of the subtree. Each Update message contains the logical location of the node that sends the message. Therefore, a rerouting node can obtain the routing of a subtree with the logical locations of the rerouting nodes in the subtree. In our protocol, the information stored at any two rerouting nodes is symmetric. A rerouting node \( m \) is located in a subtree stored at node \( n \) if and only if \( n \) is also in a subtree stored at node \( m \). In other words, rerouting node \( m \) can receive the Update message from \( n \) if and only if \( n \) can also receive the Update message from \( m \). In the following, we first present a simple way that each rerouting node individually disseminates the Update messages. We then describe an aggregation approach to reducing the number of Update messages and show that each arc in our protocol delivers at most two Update messages.

1) Individual Approach: Each satellite periodically and individually sends an Update message in each direction of the multicast tree. When a Steiner or a corner node receives the message, it copies the message and sends the copied message toward the perpendicular direction, namely right or left with respect to the original direction. The original Update message propagates in the original direction until it is copied and sent to both of the two perpendicular directions or until it reaches a corner node. Each Update message needs to be copied and sent at most once to each direction that is perpendicular to the propagation direction of the message, because the message is only required to reach the nodes in the same rerouting region. Consider Fig. 3(d), for example. Node \( k \) periodically sends an Update message toward node \( b \). When the message arrives at node \( e \), node \( e \) copies and sends the message toward node \( d \). The original message propagates until it reaches node \( b \). Node \( b \) copies and sends the message toward node \( c \). Node \( b \) will not copy and send the message toward node \( a \) even if node \( b \) has a perpendicular segment extending to the left. The reason is that each Update message can be copied and sent to each perpendicular direction at most once, and the message is copied and sent toward the left by node \( e \). Each copied message propagates until it reaches a corner or a Steiner node with a segment extending to the identical perpendicular direction. For example, in Fig. 3(a), the Update message of node \( a \) is copied and sent by node \( g \) in the segment \( s_{g,i} \) perpendicular to the left with respect to the original direction, namely segment \( s_{a,g} \). The copied message propagates \( h \) hops when it reaches node \( i \) because \( i \) also has a segment perpendicular to the left, i.e., segment \( s_{i,c} \). Therefore, node \( a \) is in the subtree stored at node \( i \).

2) Aggregation Approach: Our protocol may induce a large amount of control overhead if each rerouting node individually disseminates Update messages. Each link in this case may deliver the Update messages from multiple nearby rerouting nodes. To reduce the number of control messages on a link, our protocol aggregates multiple Update messages in the individual approach, and each Update message contains multiple Node Blocks, where each Node Block corresponds to the Update message of a rerouting node in the individual approach. Each Node Block contains the following information to propagate in the same way as the Update message in the individual approach.

1) Node ID and the Logical Location of the rerouting node adds the Node Block to the Update message.
2) Copy Flag. A Steiner or a corner node copies and sends an Update message containing the Node Block toward the perpendicular direction specified by the Copy Flag. The Copy Flag can be left (L), right (R), both (B), or none (N).
3) Remove Flag. A Steiner node removes the Node Block from the Update message if it has a segment extending to the perpendicular direction specified by the Remove Flag. The Remove Flag can be left (L), right (R), or none (N).
4) Sizes of the subtrees in the left and right rerouting regions in the forwarding direction of the Node Block. Our protocol decides the rerouting initiator of a rerouting region with the size of the subtree stored in each rerouting node.

In the aggregation approach, the root periodically sends an Update message containing its Node Block to each child node. When a rerouting node receives an Update message, it adds its Node Block and sends the message to each child node. Since each Update message propagates in the direction that at least one Node Block in the message needs to move, each Update message must propagate downstream until it reaches a leaf node, and each downstream arc delivers one Update message. After receiving the Update message, each leaf node
also sends an Update message with its Node Block upstream. After receiving the Update messages from all child nodes, each rerouting node adds its Node Block and sends the message to the parent node. Therefore, each upstream arc delivers one Update message. In addition, a rerouting node may also need to send the Update messages to some child nodes after it receives the Update messages from all child nodes. A rerouting node needs to send an Update message to a child node if at least one Node Block is required to propagate in that direction. Therefore, each downstream arc may need to deliver one more Update message. Overall, each upstream and downstream arc delivers at most one and two Update messages in our protocol, and the overhead to find the routing of subtrees for each rerouting node is limited.

In addition to the Node ID, Logical Location, Copy Flag, and Remove Flag, each Node Block also contains the sizes of the subtrees in the rerouting regions stored at the rerouting node. A rerouting node in a rerouting region may not store the complete subtree in a rerouting region. For example, node $h$ in Fig. 3(a) does not store $s_{a,b}$ in the subtree of $R_{a,g,h,c}$ because the rerouting message of $b$ does not reach $h$. However, there must exist at least one rerouting node that stores the complete subtree such as node $g$ in Fig. 3(a) because each boundary of a rerouting region can have at most one complete segment in our algorithm. Therefore, for each rerouting region, the rerouting node with the largest size of the subtree stores the complete subtree in the rerouting region, and the rerouting node acts as the rerouting initiator in our protocol. Our protocol allows the rerouting node that stores the complete subtree to initiate the rerouting procedure.

### C. Rerouting the Multicast Tree

In the rerouting procedure, a segment in a rerouting region may be created or removed. To avoid packet loss in the rerouting procedure, our protocol first creates the new segments and then removes the old segments. The rerouting initiator of a rerouting region requests some rerouting nodes in the subtree to create segments via Invite messages, which specifies the created segments. Each of these rerouting nodes then creates a segment via a Join message. The terminal node of a joining path returns a Grant message after it receives a Join message. The rerouting node sends a Leave message to the old parent node after it receives the Grant message, and the old segments are then removed. Note that the rerouting initiator may need to request a rerouting node to join a child node because the child node becomes the new parent node after the subtree is rerouted. Each node in our protocol can be involved in the rerouting procedure of at most one rerouting region at any time. Because a rerouting procedure is initiated by the Invite message sent by the rerouting initiator of a rerouting region, each node avoids participating in two rerouting procedures of two different rerouting regions by processing and forwarding only the Invite message of one rerouting initiator at each time. The node discards the Invite message from any other rerouting initiator, and the initiator will fail to receive the Join messages from neighboring nodes and will later try to reroute the multicast tree.

### V. Simulation Results

This section shows our simulation results. We use a satellite network with 24 planes and 12 satellites in each plane. We use uniform and nonuniform distributions [4] to determine the group members. In the nonuniform distribution, the root is randomly chosen among all satellites, and other group members are clustered into multiple islands. The radius of each island is uniformly distributed between $1$ and $R$. The number of members in an island is uniformly distributed between $1$ and $M$ and cannot exceed the total number of nodes in the island. The central node of each island is randomly selected among all satellites, and other members in the island are scattered around the central node. The nonuniform distribution represents the scenarios that the receivers of a multicast group are located in several locations geographically close to each other. For example, the receivers are located in the main cities of a country or a continent. We first compare our formulation with the formulations for general Steiner trees: Single_flow and Multi_flow. Single_flow and Multi_flow are the two ILP formulations in [10]. Single_flow uses a single variable for each link to describe the amount of flows on the link from the root to each member. In contrast, Multi_flow adopts an individual variable for each member to describe the amount of flows on the link from the root to the member. Both formulations find the routing of a multicast tree according to the routing of the flow sent to all members. We use a personal computer with an Intel Core 2 Duo 3.0-GHz processor running CPLEX 9.0 to solve the ILP problem. Due to the large computational time of solving ILP problems, we only consider multicast trees with no more than 50 members. Each member is uniformly determined among all logical locations. For the proposed distributed algorithm in the simulation, we choose a member to join an existing multicast tree and reroute the tree at each iteration. The simulation iteratively adds the members until the number of members reaches the limit, and we then compare the obtained tree with the optimal solution.

Fig. 6 shows the computational time of all formulations to find the optimal solutions. Our formulation is at least ten times faster than the other two formulations because we use fewer variables and constraints and have smaller search space. Fig. 7 compares the solution obtained by our distributed
Fig. 7. Comparisons of the multicast trees created by our algorithm, the lower bound on our formulation (Rect_flow), and the lower bounds on the formulations for general Steiner trees (Single_flow and Multi_flow).

Fig. 8. Ratios of the total length of a multicast tree created by the shortest path algorithm (SP) and our algorithm (Sliding) to the total length of an RST.

Algorithm with the lower bound on the optimal solution. The lower bound associated with each formulation is the optimal solution to its linear relaxation problem, i.e., linear relaxation on the formulation. Fig. 7 shows that our formulation is tighter than the other two formulations. Both of the solution obtained by our distributed algorithm and the lower bound associated with our formulation are very close to the optimal solution. Fig. 7 also explains that the computational time of formulation Single_flow in Fig. 6 is very large. The loose lower bound of Single_flow in Fig. 7 indicates that the constraints of Single_flow are not tight in its linear relaxation problem. Therefore, the branch-and-cut algorithm used by CPLEX has to branch much more nodes in the search tree to find the optimal solution. The difference of our solution and the optimal solution is within 5% of the optimal solution for all group sizes. Fig. 9 compares the shortest path algorithm and our algorithm for the multicast trees with more group members. The shortest path algorithm finds the minimum-hop path between the root and each member. If there exists multiple minimum-hop paths for some members, the members select the paths that share more common links to aggregate the path and reduce the number of links in a tree. Compared with the shortest path algorithm, our algorithm saves about 40% network bandwidth for the uniform distribution of members and about 55% for the nonuniform distribution of members. The multicast trees with the nonuniform distribution of members are smaller than the trees with the uniform distribution of members since members in the former case tend to cluster together and share the same path to the root.

Fig. 10 shows the sum of the control messages sent on each link in each update and rerouting procedures. As the group size increases, more links are involved in delivering Update messages. However, the size of a rerouting region decreases, and fewer Invite, Join, Grant, and Leave messages are sent in a rerouting region during a rerouting procedure. As the group size increases, the decrement of the overhead in a rerouting procedure is reduced because more Steiner nodes, corner nodes, and members are in a rerouting region. As $M$ increases, group members tend to cluster together, and the size of a multicast tree and that of a rerouting region become smaller. Our protocol thereby induces less overhead.

Fig. 11 shows the total length of a multicast tree after each update and rerouting procedures. The group members are uniformly distributed among all satellites. Our algorithm can reduce about 30% of the bandwidth cost of an original unrerouted multicast tree. The unrerouted multicast tree is not the shortest path tree, and the tree is constructed by each member that joins the closest on-tree member at Step 1 in Fig. 5. Fig. 11 also shows that the reduction in tree length is mostly achieved in the first few procedures.

Fig. 12 compares the trees for the support of dynamic group membership with and without our rerouting procedure. Initially, we pick a multicast tree containing members randomly distributed among all satellites. Then, we choose a new member (or an existing member) to join (or leave) the multicast group, and this procedure is repeated. Fig. 12(a) shows the results after a certain number of members (specified in the parentheses) have joined or left. The results without rerouting, i.e., the three curves labeled WOR($\cdot$), indicate that the multicast tree deteriorates as more members join or leave. After the specified number of members joins or leaves, we reroute the multicast tree, and the results, i.e., the three curves labeled WR($\cdot$), show that our algorithm can significantly reduce the tree length. In addition, the small difference between WR(10) and WR(30) indicates that our algorithm is still effective after a large number

Authorized licensed use limited to: National Taiwan University. Downloaded on January 20, 2009 at 23:29 from IEEE Xplore. Restrictions apply.
of members have joined or left the group. Fig. 12(b) shows the tree length after each member joins or leaves the multicast group. Initially, the tree contains 15 members. The multicast tree deteriorates if we do not perform the rerouting procedure. However, if we reroute the tree after every 18 members join or leave, the tree length drops. Therefore, the results show that our rerouting algorithm can effectively limit the deterioration of the multicast tree and is capable of supporting dynamic group membership.

In summary, our protocol induces a limited overhead in rerouting procedures even when the group size is large, where the overhead is the product of the number of rerouting procedures and the sum of control messages sent on each link in each rerouting procedure. According to Figs. 6 and 10, our protocol uses each link in a multicast tree to deliver several control messages to reduce the tree length. For the update procedure, although each link in a multicast tree needs to deliver slightly more Update messages, the results in Fig. 11(a)
show that only the first few update procedures can reduce most of the bandwidth. Therefore, we can limit the number of update procedures to reduce the overhead in our protocol.

VI. CONCLUSION

In this paper, we propose a bandwidth-efficient multicast routing mechanism using RSTs for IP-based LEO satellite networks. We propose a new ILP formulation for this problem. Our formulation requires much less computational time than the formulations for general Steiner trees. We also design a distributed algorithm and protocol that supports dynamic group membership. Our simulation results show that the multicast trees generated by our algorithm use 40% less bandwidth than the shortest path trees. The difference between the optimal solutions and the solutions obtained by our algorithm is within 5%. Moreover, our simulation results show that the overhead induced by our protocol is limited, and our protocol scales to a large number of group members.

REFERENCES