# A Distributed Protocol to Serve Dynamic Groups for Peer-to-Peer Streaming

Xing Jin, S.-H. Gary Chan, *Senior Member*, *IEEE Computer Society*, Wan-Ching Wong, and Ali C. Begen, *Member*, *IEEE Computer Society*

**Abstract**—Peer-to-peer (P2P) streaming has been widely deployed over the Internet. A streaming system usually has multiple channels, and peers may form multiple groups for content distribution. In this paper, we propose a distributed overlay framework (called *SMesh*) for dynamic groups where users may frequently hop from one group to another while the total pool of users remain stable. SMesh first builds a relatively stable mesh consisting of all hosts for control messaging. The mesh supports dynamic host joining and leaving, and will guide the construction of delivery trees. Using the Delaunay Triangulation (DT) protocol as an example, we show how to construct an efficient mesh with low maintenance cost. We further study various tree construction mechanisms based on the mesh, including embedded, bypass, and intermediate trees. Through simulations on Internet-like topologies, we show that SMesh achieves low delay and low link stress.

**Index Terms**—Peer-to-peer streaming, dynamic group, Delaunay triangulation.

## 1 Introduction

With the penetration of broadband Internet access, there has been an increasing interest in media streaming services. Recently, P2P streaming has been proposed and developed to overcome the limitations of traditional server-based streaming. In a P2P streaming system, cooperative peers self-organize themselves into an overlay network via unicast connections. They cache and relay data for each other, thereby eliminating the need for resourceful servers from the system. Today, several practical P2P streaming software implementations have been shown to be able to serve up to thousands of peers with acceptable quality of service [1], [2], [3].

In a P2P streaming system, the server (or a set of servers) usually provides multiple channels. A peer can freely switch from one channel to another. For example, one of the most popular P2P streaming systems, PPLive, has provided over 400 channels in September 2007 [4]. According to a measurement study from the Polytechnic University, the total number of peers in PPLive during a day in 2007 varies from around 50 thousand to 400 thousand, and the number of peers in a single channel, e.g., CCTV1, varies from several hundred to several thousand [5], [6]. We can see that there is a large pool of peers in the streaming network. Peers are divided into multiple small groups, each corresponding to a channel. Peers in the same group share and relay the same streaming content for each other. In another study, a six-month 150-channel IPTV trace shows that people frequently change from one channel to another, with the median and mean channel holding time being 8 seconds and 14.8 minutes, respectively [7]. The trace also shows that a household (among those watching TV for the longest time) on average watches 2.54 hours and 6.3 distinct channels of TV a day.

In fact, there are many other similar applications over the Internet. In the application, the system contains multiple groups with different sources and contents. A user may join a specific group according to its interest. While the lifetime of users in the system is relatively long and the user pool is rather stable, users may hop from one group to another quite frequently. Examples include stock quotes, news-on-demand, and multisession conferencing. A more typical example is group chat of Skype [8]. Skype allows up to around 100 users to chat together. While millions of Skype users stay online and relay data for each other, the users may form multiple small groups for group chat. According to Rossi et al. [9], except for very short sessions, most Skype peers are alive for about one third of a day. Generally, such lifetime of a Skype peer is longer than the duration of a group chat.

In above applications, as peers may dynamically hop from one group to another, it becomes an important issue to efficiently deliver specific contents to peers. One obvious approach is to broadcast all contents to all hosts and let them select the contents. Clearly, this is not efficient in terms of bandwidth and end-to-end delay, especially for unpopular channels. Maintaining a separate and distinct delivery overlay for each channel appears to be another solution. However, this approach introduces high control overhead to maintain multiple dynamic overlays. When users frequently hop from one channel to another, overlay reformation becomes costly and may lead to high packet loss.

In this paper, we consider building a data delivery tree for each group. To reduce tree construction and maintenance costs, we build a single shared overlay mesh. The mesh is formed by all peers in the system and is, hence, independent

- *X. Jin is with the Systems Technology Group, Oracle USA, Inc., 400 Oracle Parkway, Redwood Shores, CA 94065. E-mail: xing.jin@oracle.com.*
- *S.-H.G. Chan and W.-C. Wong are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR. E-mail: gchan@cse.ust.hk.*
- *A.C. Begen is with Video and Content Platforms Research and Advanced Development Group, Cisco Systems, Inc., San Jose, CA 95134. E-mail: abegen@cisco.com.*

*Manuscript received 22 Jan. 2008; revised 22 Feb. 2009; accepted 2 Mar. 2009; published online 26 Mar. 2009.*
*Recommended for acceptance by P. Mohapatra.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-01-0026.*
*Digital Object Identifier no. 10.1109/TPDS.2009.58.*

of joining and leaving events in any group. This relatively stable mesh is used for control messaging and guiding the construction of overlay trees. With the help of the mesh, trees can be efficiently constructed with no need of loop detection and elimination. Since an overlay tree serves only a subset of peers in the network, we term this framework *Subset-Mesh*, or *SMesh*.

Our framework may use any existing mesh-based overlay network. In this paper, we use Delaunay Triangulation (DT) as an example [10]. We propose several techniques to improve the DT mesh, e.g., for accurately estimating host locations and distributed partition detection. Based on the mesh, we study several tree construction mechanisms to trade off delay and network resource consumption. We investigate the following two important issues in SMesh:

- *Mesh formation and maintenance*: The mesh should be efficiently formed and maintained in order to reduce control and delivery overhead. We choose the DT mesh as an example. We then use the Global Network Positioning tool (GNP) [11] (or many other equally good ones [12], [13], [14]) to estimate host locations in the Internet in order to improve mesh efficiency. We further present a distributed algorithm on how to detect and recover mesh partitions. Our mesh has the following properties:
    - Low delivery delay: As mesh formation and message forwarding are based on hosts' network locations, the delay for data delivery is significantly reduced as compared to that in the traditional DT mesh.
    - Distributed: Unlike the traditional DT mesh, SMesh does not require a central server for mesh maintenance. It is fully distributed and scalable.
- *Construction of data delivery trees*: Given the mesh, we study how source-specific overlay trees can be efficiently constructed and maintained. We consider three ways to construct a tree: 1) Embedded tree, where tree branches are all mesh edges; 2) Bypass tree, where tree nodes can only be group members and tree branches may not be mesh edges; and 3) Intermediate tree, which is a trade-off between an embedded tree and a bypass tree. These trees have the following properties:
    - Overhead reduction: As compared to traditional tree-based protocols, SMesh achieves much lower control overhead for tree construction and maintenance. This is because the mesh has maintained enough host information and can efficiently deal with host hopping between different groups.
    - QoS provisioning: SMesh provides QoS in the following senses: 1) It limits the node stress of a host in a tree according to the host's capability. 2) It aggregates long-delay paths and delegates data delivery to shorter paths. As a result, packets may take more hops to reach their destinations, and this trades off end-to-end delay with network resource consumption.

SMesh does not rely on a static mesh. In the case of host joining or leaving, the underlying DT mesh can automatically adjust itself to form a new mesh. The trees on top of it will then accordingly adjust tree nodes and tree edges. Also note that in SMesh a host may join as many groups as its local resource allows. If a host joins multiple groups, its operations in different groups are independent of each other.

The rest of the paper is organized as follows: In Section 2, we review the GNP and DT protocols. In Section 3, we discuss how to form and maintain an efficient mesh. In Section 4, we study the tree construction mechanisms. In Section 5, we present illustrative simulation results on Internet-like topologies. We discuss related work in Section 6, and finally conclude the paper in Section 7.

## 2 Review on GNP and DT

In this section, we briefly review the GNP and DT protocols. We will construct our SMesh system based on these building blocks.

### 2.1 Review on GNP Estimation

GNP estimates host coordinates in a multidimensional euclidean space such that the distance between two hosts in the euclidean space correlates well with the measured round-trip time between them [11]. In GNP, a few hosts are used as landmarks. Landmarks first measure the round-trip time between each other and forward results to one of them. The landmark receiving results uses the results to compute the landmark coordinates in the euclidean space. The coordinates are then disseminated back to the respective landmarks. More specifically, to estimate landmark coordinates, the following objective function is minimized[1]:

$$J_{landmark}(L_1, L_2, \ldots, L_M) = \sum_{L_i, L_j \in \{L_1, \ldots, L_M\} | i > j} \left( \|L_i - L_j\| - RTT(i,j) \right)^2, \qquad (1)$$

where $M$ is the number of landmarks, $L_i$ and $L_j$ are the coordinates of landmarks $i$ and $j$ in the euclidean space, and $RTT(i,j)$ is the round-trip time between $i$ and $j$. As shown, $J_{landmark}$ is the sum of the estimation error between the measured round-trip time and the logical distances in the euclidean space among the landmarks. Therefore, we seek a set of landmark coordinates such that the sum is minimized. If there are multiple sets of $\{L_1, L_1, \ldots, L_M\}$ to minimize $J_{landmark}$, any one set can be used.

Given the landmark coordinates, a normal host estimates its coordinates by minimizing a similar objective function:

$$J_{host}(H_u) = \sum_{L_i \in \{L_1, \ldots, L_M\}} \left( \|H_u - L_i\| - RTT(u,i) \right)^2, \qquad (2)$$

where $H_u$ is the coordinates of host $u$, and $RTT(u,i)$ is the measured round-trip time between host $u$ and landmark $i$.

Note that landmarks do not have to be permanent. It is easy to modify (2) to remove a failed landmark or add a new landmark. According to Ng and Zhang [11], GNP has shown good enough performance when the number of landmarks is a constant (9-15 as recommended in [11] and 20 in our simulations). Therefore, each host can obtain its coordinates

1. Equations (1) and (2) show the cases in two-dimensional space as originally proposed in [11].

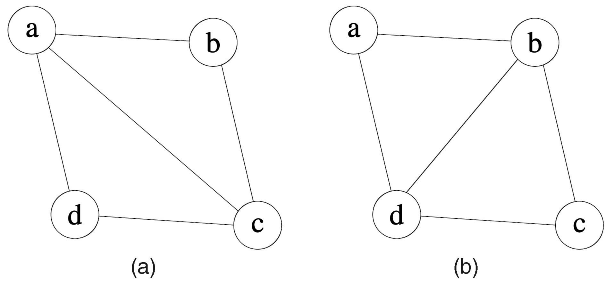


Fig. 1. (a) Two adjacent triangles in a convex quadrilateral ($\triangle abc$ and $\triangle adc$) violate the DT property and (b) restore the DT property by disconnecting $a$ from $c$ and connecting $b$ and $d$.

by pinging $O(1)$ landmarks and using $O(1)$ messages. It is highly efficient and scalable.

## 2.2 Review on Delaunay Triangulation

In the traditional DT protocol, each host knows its geographic coordinates [10]. Hosts form a DT mesh based on their geographic coordinates. Compass routing, a kind of local routing, is then used to route a message along the mesh [15]. In this approach, a host only needs to know the states of its immediate neighbors to construct and maintain the mesh, and the mesh is adaptive to dynamic host joining or leaving.

DT protocol connects hosts together so that the mesh satisfies the DT property, i.e., the minimum internal angle of the triangles in the mesh is maximized [16], [17]. Here angles are computed according to the coordinates of hosts as in traditional geometry. It has been shown that a mesh formed in this way connects close hosts together. We illustrate the triangulation process in Fig. 1. Suppose that hosts $a, b, c$, and $d$ form a convex quadrilateral $abcd$. Two possible ways to triangulate it are shown in Figs. 1a and 1b, respectively. Clearly, the minimum internal angle of $\triangle abc$ and $\triangle acd$ is smaller than that of $\triangle abd$ and $\triangle bcd$. DT protocol then transforms the former configuration into the latter one. To achieve this, a host periodically sends *HelloNeighbor* messages to its neighbors to exchange their neighborhood information. It removes a host from its neighbor list if the connection to that host violates the DT property. Similarly, a host adds another host into its neighbor list only if the addition does not violate the DT property. Given a set of $N$ hosts in the network, a DT mesh among them can be constructed with $O(N \log N)$ messages. The detailed construction mechanism and complexity analysis can be found in [16].

Compass routing works as follows. Host $a$ forwards messages with destination $t$ to $b$, if $b$, among all $a$'s neighbors, forms the smallest angle to $t$ at $a$. We show an example in Fig. 2. Hosts $b, c$, and $d$ are neighbors of host $a$, and $a$ needs to forward a message to destination $t$. Since $\angle bat$ is the smallest among $\angle bat, \angle cat$, and $\angle dat, b$ is chosen by $a$ as the next hop for message forwarding.

In the traditional DT protocol, mesh partition is detected by a central server. In each connected DT mesh, a host is selected as the leader, which periodically exchanges control messages with the server. If the mesh is partitioned, more than one host will claim to be leaders. The server then requests them to connect to each other.

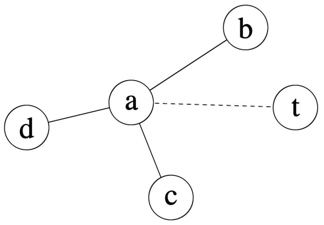


Fig. 2. An example of compass routing. The angle $\angle bat$ is smaller than angles $\angle cat$ and $\angle dat$. Therefore, when $a$ receives a message with destination $t$, it forwards the message to $b$.

In summary, the traditional DT protocol has the following limitations:

- Inaccuracy in estimating host locations: DT estimates host locations based on their geographic coordinates. This may work well for wireless networks, but not for the Internet where the network delay between two hosts may not correlate well with their geographic distance. In other words, the traditional DT protocol may build a tree with a low geographic distance but a high end-to-end network delay.
- Single point of failure: Partition detection and recovery rely on a central server. This forms a single point of failure and is not scalable.
- Message looping: Compass routing in connected DT mesh does not result in loops [10], [15]. However, looping may persist in partitioned meshes. If the destination of a message is in another mesh, the message may loop in the current mesh for a long time.

## 3 Mesh Formation and Maintenance

In this section, we discuss how SMesh forms and maintains an efficient mesh. SMesh addresses the above problems as follows:

- SMesh uses GNP to estimate host locations in the Internet space and builds a DT mesh based on the estimated host coordinates. Since GNP estimation is based on network distances between hosts, the resultant mesh can achieve lower end-to-end delay than the traditional DT mesh.
- SMesh uses a distributed algorithm to detect and recover mesh partition, thereby eliminating the need for a central server from the system.
- The distributed algorithm is able to detect whether a message destination is in a partitioned mesh or not, and hence solves the message looping problem.

### 3.1 Distributed Algorithm for Partition Detection and Recovery

We now present a distributed algorithm to detect and recover mesh partition. We define some notations as follows. Given a graph, define $\angle_+ abc$ as the clockwise angle from edge $ab$ to edge $bc$ and $\angle_- abc$ as the counter-clockwise angle from edge $ab$ to edge $bc$. They are both between 0 degree and 360 degrees (i.e., the angle is not negative). We further define the undirected angle $\angle abc$ as the smaller one of $\angle_+ abc$ and $\angle_- abc$, which is certainly between 0 degree and 180 degrees. We show the examples of $\angle_+, \angle_-$, and $\angle$ in Figs. 3a, 3b, and 3c, respectively.

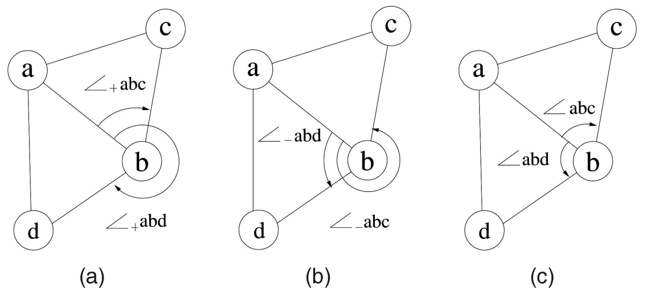


Fig. 3. Examples of (a) clockwise angles $\angle_{+}$, (b) counterclockwise angles $\angle_{-}$, and (c) undirected angles $\angle$.

We further consider two connected hosts $b$ and $c$, and another host $a$ in the graph (whether $a$ is a neighbor of $b$ or $c$ is irrelevant here). We say that $c$ is the clockwise neighbor of $b$ with respect to $a$ if and only if $\angle_{+}abc$ is less than 180 degrees and is the minimum among all the neighbors of $b$ (i.e., $\angle_{+}abc \leq \angle_{+}abx, \forall x \in$ neighbors of $b$). In this case, we write $N^{+}_{b,a} = c$ (one can imagine that the edge $ba$ with $b$ fixed, when sweeping clockwise by less than 180 degrees, would first touch $c$ among all $b$'s neighbors). For example, in Fig. 3a, $c$ is the clockwise neighbor of $b$ with respect to $a$ (i.e., $N^{+}_{b,a} = c$). Similarly, we say that $c$ is the counterclockwise neighbor of $b$ with respect to $a$, denoted as $N^{-}_{b,a} = c$, if and only if $\angle_{-}abc$ is less than 180 degrees and is the minimum among all the neighbors of $b$. In Fig. 3b, $d$ is the counterclockwise neighbor of $b$ with respect to $a$ (i.e., $N^{-}_{b,a} = d$). Note that host $b$ may not have any clockwise neighbor (or counterclockwise neighbor) with respect to $a$. For example, in Fig. 3, host $b$ does not have any clockwise neighbor with respect to $c$, since angles $\angle_{+}cbd$ and $\angle_{+}cba$ are larger than 180 degrees.

**Theorem 1.** *Given the above definitions and host coordinates, a host $u$ detects that a destination $t$ is partitioned from the mesh if and only if one of the following conditions is satisfied:*

1. $N^{+}_{u,t} = \emptyset$, *or*
2. $N^{-}_{u,t} = \emptyset$, *or*
3. $\angle_{+}N^{-}_{u,t}uN^{+}_{u,t} > 180°$, *or*
4. $\angle_{+}N^{+}_{u,t}tN^{-}_{u,t} > 180°$.

**Proof.** There are two possible cases for $t$'s location:

- $t$ is outside the mesh (see Figs. 4a, 4b, and 4c).

  By definition, a DT mesh is a convex polyhedron where only the external angles are larger than 180 degrees. As $t$ falls outside the mesh, a message with destination $t$ must be finally forwarded to a boundary host $u$ in the mesh. The possible positions of $t$ are given in Figs. 4a, 4b, and 4c, where hosts $b$ and $c$ are two neighbors of $u$ on the boundary of the mesh. Fig. 4a corresponds to condition 1. Fig. 4b corresponds to condition 2. Fig. 4c corresponds to condition 3, where $\angle_{+}N^{-}_{u,t}uN^{+}_{u,t}$ is as indicated.
- $t$ is in the interior of the mesh (see Fig. 4d).

  If $t$ is in the interior of the mesh, the position of $t$ must fall inside a certain triangle $\triangle ubc$ (as shown in Fig. 4d). When a host $u$ receives a message with destination $t$, if it finds that $\angle_{+}N^{+}_{u,t}tN^{-}_{u,t} > 180$ degrees and there is no connection with $t$, it can conclude that $t$ is not in the mesh. □

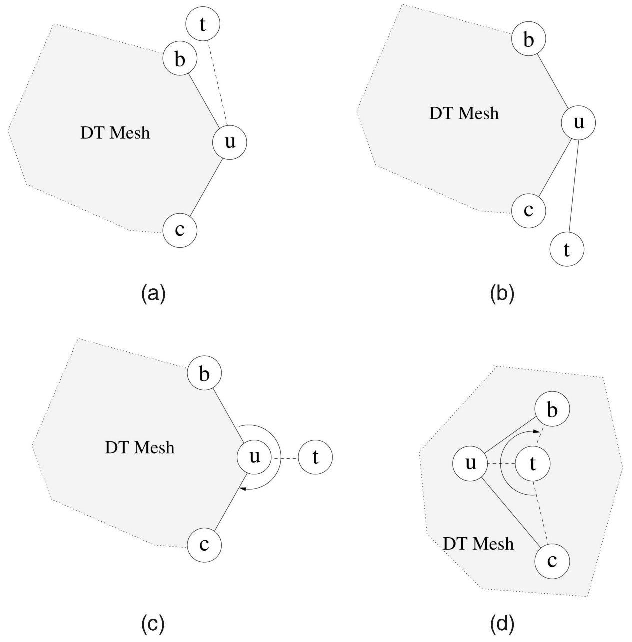


Fig. 4. (a), (b), and (c) Host $u$ is on the boundary of the overlay mesh, and host $t$ lies outside the mesh. (d) $u$ is an interior host of the mesh. Host $t$ lies inside triangle $\triangle ubc$, but does not belong to the mesh.

Therefore, a host $u$ checks whether the destination has been partitioned from its mesh before forwarding a message. If so, $u$ directly forwards the message to $t$ to avoid message looping, and asks $t$ to join the mesh through itself (using the joining mechanism below) so as to recover the partition.

## 3.2 Joining Mechanism

A joining host, after obtaining its coordinates, sends a *MeshJoin* message with its coordinates to any host in the system. *MeshJoin* is then sent back to the joining host along the DT mesh based on compass routing. Since the joining host is not a member of the mesh yet, it can be considered as a partitioned mesh consisting of a single host. The *MeshJoin* message finally triggers the partition recovery mechanism at a particular host in the mesh, which helps the new host join the mesh.

We illustrate the host joining mechanism in Fig. 5. Suppose that $u$ is a joining host. The following steps show how $u$ joins the mesh (corresponding to Fig. 5):

1. $u$ first retrieves the list of landmarks by querying a host $b$ with a *GetLandmark* message.
2. Then $u$ measures the round-trip time to the landmarks and estimates its coordinates.
3. After that, $u$ sends a *MeshJoin* message to $b$.
4. The message is then forwarded from $b$ to $c$ based on compass routing.
5. Since $u$ falls into $\triangle acd$, $c$ knows that $u$ is in another partitioned mesh. $c$ then adds $u$ into its neighbor list $N_c$ to recover the partition. Note that the minimum internal angle of $\triangle auc$ and $\triangle abc$ is less than that of $\triangle buc$ and $\triangle abu$. Therefore, the connection between $c$ and $a$ violates the DT property, and $c$ will remove $a$

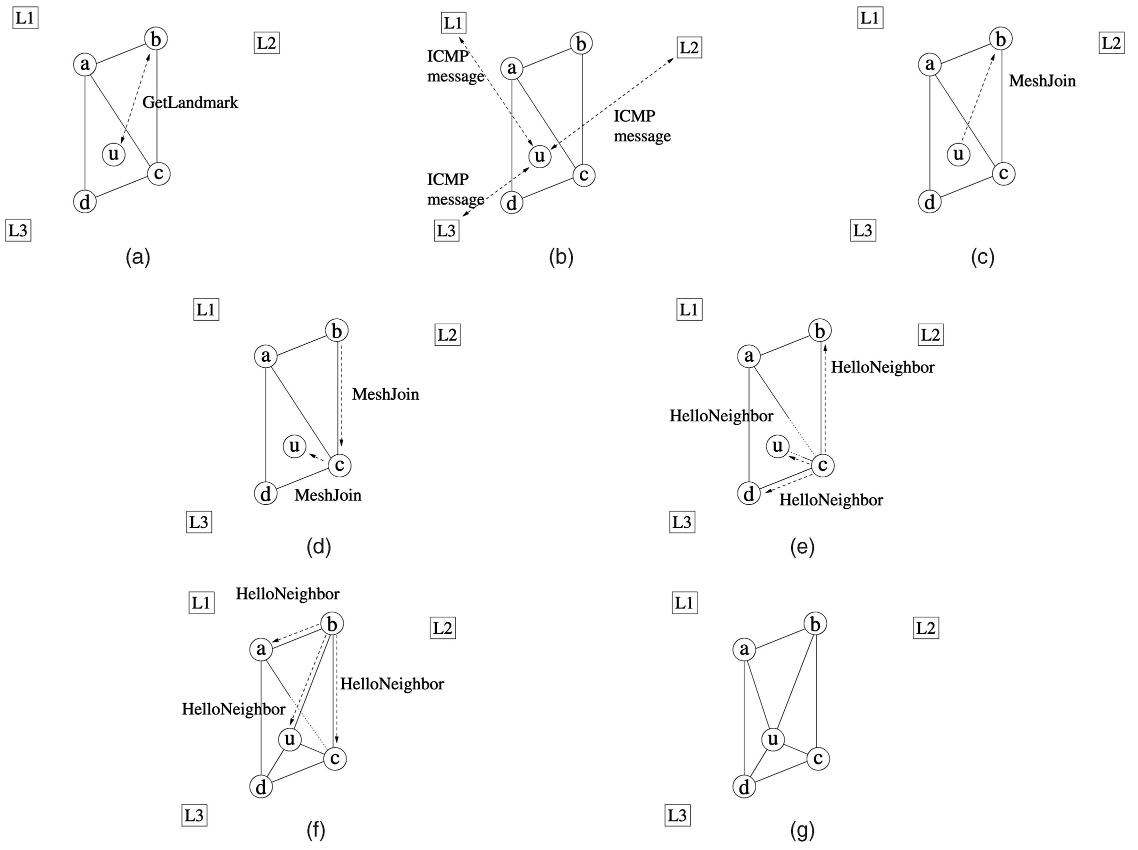


Fig. 5. An example of host joining in SMesh.

from $N_c$ and notify $a$ to remove the connection. $c$ then broadcasts its neighborhood information to its neighbors through *HelloNeighbor* messages. (In DT, each host needs to periodically send *HelloNeighbor* messages to its neighbors to exchange the neighborhood information.)

6. Upon receiving *HelloNeighbor* messages from $c, b$, and $d$ discover $u$. They add $u$ into their neighbor lists since such connections do not violate the DT property. In the meantime, $u$ also discovers $b$ and $d$ and adds them into its neighbor list. Suppose that $b$ is the next to broadcast *HelloNeighbor* messages. Upon receiving the message, $a$ discovers $u$ and adds $u$ into its neighbor list.
7. The resultant overlay mesh after the joining of $u$ still satisfies the DT property.

## 4 Construction of Data Delivery Trees

In this section, we discuss tree construction mechanisms in SMesh. In Section 4.1, we propose three algorithms to construct data delivery trees on top of the mesh. In Section 4.2, we present a path aggregation algorithm for QoS provisioning. In Section 4.3, we illustrate our algorithms with examples.

### 4.1 Embedded, Bypass, and Intermediate Trees

We study three ways to build trees in SMesh. The first type of tree is called an *embedded tree*, where all tree edges are part of the overlay mesh. When forming the tree, nonmember hosts may be included. This is similar to Skype routing, where a Skype client may help relay packets that it is not interested in. The second one builds an overlay tree that covers only group members without having to use mesh edges. We call it a *bypass tree*. All tree nodes in a bypass tree are members of the group. This is similar to traditional overlay tree construction, where a node relays packets only for other members in its groups. However, the construction of a bypass tree has to rely on the underlying mesh. The third one is termed an *intermediate tree*, which lies between embedded and bypass trees. In the following, we call a nonleaf host in an overlay tree a *forwarder*, which needs to forward data messages to its children in the tree. We elaborate the details as follows:

- *Embedded Tree*: To join an embedded tree, a joining host first sends a *TreeJoin* message to the group source along the DT mesh using compass routing. All hosts along the message routing path become forwarders for the tree no matter whether they are group members.

  In Algorithm 1, we show how the *TreeJoin* message is handled by a host in the mesh: A host first adds the joining host into its children table for the specified group. Then, it checks whether itself is already a forwarder of the group. If so, the host has already started to forward messages for the tree and known the overlay path along the mesh to the group source. So, it suppresses the forwarding of the *TreeJoin* message and does nothing. Otherwise, it turns itself into a forwarder and relays the *TreeJoin* message to the group source. The *TreeJoin* message will eventually discover a path along the mesh to the group source.

- *Bypass Tree*: All forwarders in a bypass tree are the group members. Similarly, in order to join a bypass tree, a joining host needs to send a *TreeJoin* message to the group source using compass routing. We show the tree construction algorithm in Algorithm 2. A nonmember host receiving the *TreeJoin* message simply relays the message to the next hop without turning itself into a forwarder. Such a host will not forward data packets for the group in the future. On the other hand, if the host receiving the message is a member of the group, it accepts the joining host as its child by adding the joining host into its children table. Clearly, such a host has already joined the tree and known the path to the group source. So, it stops forwarding the *TreeJoin* message.
- *Intermediate Tree*: We observe that an embedded tree requires the participation of nonmember hosts, and a host may need to serve multiple hosts of different groups. As compared to a bypass tree, it consumes more network resources and suffers from higher delay, especially for sparse groups. On the other hand, a host in a bypass tree may have a high node stress and heavy load for data forwarding (e.g., a star-like topology rooted at the source for a sparse group). Therefore, we propose an intermediate tree which trades off between an embedded tree and a bypass tree. In an intermediate tree, a nonmember host is included in the tree if it receives more than a certain number of joining messages. Such a host resides in many routing paths, and we expect high delivery efficiency by including it in the tree. In Algorithm 3, we show how the *TreeJoin* message is handled by a host: A host handles the message as in a bypass tree if the number of received messages is less than a certain threshold. Otherwise, the host forwards the message as in an embedded tree.

**Algorithm 1.**
```
TREEJOINHANDLER_EMBEDDEDTREE (TreeJoin)
1 Me.Child[TreeJoin.InterestGroup] $\leftarrow$ Me.Child[TreeJoin.
  InterestGroup] $\cup$ TreeJoin.JoinHost
2 if TreeJoin.InterestGroup $\notin$ Me.InterestGroups
3     then Me.InterestGroups $\leftarrow$ Me.InterestGroups $\cup$
             TreeJoin.InterestGroup
4          TreeJoin.JoinHost $\leftarrow$ Me
5          CompassRoute(TreeJoin, TreeJoin.GroupSource);
```

**Algorithm 2.**
```
TREEJOINHANDLER_BYPASSTREE (TreeJoin)
1 if TreeJoin.InterestGroup $\in$ Me.InterestGroups
2   then Me.Child[TreeJoin.InterestGroup] $\leftarrow$
    Me.Child[TreeJoin.InterestGroup] $\cup$ TreeJoin.JoinHost
3   else CompassRoute(TreeJoin, TreeJoin.GroupSource);
```

**Algorithm 3.**
```
TREEJOINHANDLER_INTERMEDIATETREE (TreeJoin)
1 Received Message $\leftarrow$ Received Message + 1
2 if Received Message $\leq$ Message Threshold
3     then TreeJoinHandler_BypassTree (TreeJoin);
4     else TreeJoinHandler_EmbeddedTree (TreeJoin);
```

We give three illustrative examples of the trees in Section 4.3. Please refer to it for details. Note that the above overlay trees are inherently loop free. This is because compass routing in DT is a greedy algorithm, where the distance from a host to the message destination strictly decreases along the path [15]. As a result, in a data delivery tree, the distance from the source to a host is always smaller than the distance from the source to any of its descendants. This property leads to the loop-free characteristic of SMesh trees.

One possible issue of bypass tree is that a host may have a high node stress by having many children. This is also an issue for intermediate tree. As a comparison, an embedded tree does not have this issue. Because all edges of an embedded tree are part of the mesh, while in a DT mesh, a host has on average six neighbors. In order to address this issue in bypass and intermediate trees, it is possible to set a degree bound for each host. If the number of children of a host reaches its degree bound, the host will not accept new joining hosts as its children. Instead, it forwards new joining hosts to other hosts in the tree (e.g., its parent). Clearly, this is a trade-off between fan-out and performance. It may incur a higher delay.

## 4.2 Path Aggregation for QoS Provisioning

We note that the traditional DT protocol may result in high network resource consumption. For example, if host $a$ belongs to domain $A$, and hosts $b$ and $b'$ belong to domain $B$, usually the delays of interdomain paths $ab$ and $ab'$ are much higher than that of intradomain path $bb'$. In other words, angle $\angle bab'$ is small. As a result, using compass routing, if either $b$ or $b'$ is a child of $a$, the other one is also likely to be a child of $a$. Therefore, two independent connections across domains $A$ and $B$ are set up, which leads to high usage of long paths and hence high network resource consumption. Furthermore, in the traditional DT protocol, a host may have many children. However, a host often has a node stress threshold $K$ for each group depending on its resource. To address these problems, we require that the minimum adjacent angle between two children of a host should exceed a certain threshold $T$. If the condition on $K$ or $T$ is violated, SMesh modifies its overlay tree through aggregation and delegation.

Consider a source $s$ and a host $u$ in the network. Once $u$ accepts a child, $u$ checks whether its node stress exceeds $K$ or whether the minimum adjacent angle between its children is less than $T$. If so, it runs the path aggregation algorithm, as shown in Algorithm 4. It selects a pair of children with the minimum adjacent angle and delegates the child farther from the source to the other. Note that after aggregation, the overlay tree is still loop free because hosts are still topologically sorted according to their distances from the source. We show an example in Fig. 6, where $\angle buc$ is the smallest angle among all $u$'s children. If it is smaller than the threshold $T$ and because $\|s-b\| < \|s-c\|, u$ delegates $c$ to $b$.

**Algorithm 4.**
```
PATHAGGREGATION($u$)
1   $[c, c'] \leftarrow$ a pair of children with the minimum adjacent
    angle
2   while $\angle cuc' < T$ OR number of children $> K$
3   do if $\|c - s\| < \|c' - s\|$
4        then delegate $c'$ to $c$
5        else elegate $c$ to $c'$
6      $[c, c'] \leftarrow$ a pair of children with the minimum
       adjacent angle.
```

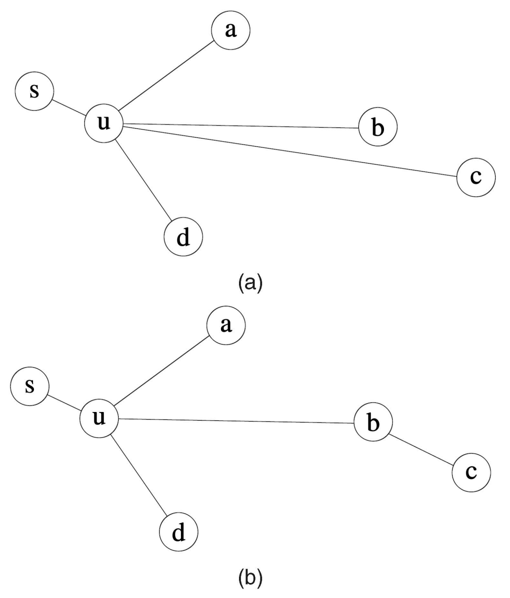


Fig. 6. Host $u$ delegates its child $c$ to child $b$, since $\angle buc < T$ and $\|s-b\| < \|s-c\|$.

SMesh avoids tree partition during aggregation by temporarily setting up backup paths. If $u$ delegates its child $c$ to another child $c'$, $u$ would keep forwarding data to $c$ unless it receives an acknowledgment from $c'$. (This way, a backup path $uc$ is set up.) Backup paths are also present when a host leaves its group. For example, in a bypass tree, a leaving host $u$ sends a *TreeLeave* request to its parent $p$ with the information of its children $C = \{c_1, c_2, \ldots, c_M\}$. $p$ keeps forwarding data to $u$ until $p$ has handled (either accepted or delegated) all the hosts in $C$ (i.e., backup paths $uc_1, uc_2, \ldots, uc_M$ are set up).

### 4.3 Illustrative Examples

We show in Figs. 7, 8, and 9 how embedded, bypass, and intermediate trees are constructed. White circles in figures denote hosts belonging to the same group. The joining sequence is $\{d, b, f, c\}$, and $s$ is the source.

We first show the construction of an embedded tree in Fig. 7. When $d$ joins, its *TreeJoin* message is first forwarded to $c$ (Fig. 7a). Although $c$ is a nonmember, the message turns it into a forwarder and it relays the message to $s$. Next, $b$ joins the group and its *TreeJoin* message is also forwarded to $c$ (Fig. 7b). Since $c$ is a forwarder already, it only modifies its children table and does not relay the message again. Afterwards, when $f$ joins, its *TreeJoin* message is first forwarded to $e$, and then to $s$ (Fig. 7c). Finally, when $c$ joins the group, it does not need to send a *TreeJoin* message since it is already a forwarder (Fig. 7d).

In Fig. 8, we show a bypass tree with forwarding delegation ($K = 2$ without $T$ threshold, i.e., $T =$ 360 degrees). When $d$ joins, its *TreeJoin* message is first forwarded to $c$ and then to $s$ (Fig. 8a). Since $c$ is a nonmember, $s$ directly serves $d$. Later on, when $b$ joins, its *TreeJoin* message is also forwarded to $s$ through $c$. Similarly, $s$ directly serves $b$ (Fig. 8b). Note that at this moment, $s$ has already had $K$ children ($K = 2$). Therefore, when $f$ joins and becomes $s$'s child, $s$ has to delegate one of its children ($d$, $b$, and $f$) to others. It first selects a pair of children with the minimum adjacent angle, which are $b$ and $d$. Then, it delegates the child farther from the source (i.e., $b$) to the other one, i.e., $d$ (Fig. 8c). Finally, when $c$ joins, $s$ similarly delegates $d$ to $c$ (Fig. 8d).

We finally show an intermediate tree with $K = 2$ and $MessageThreshold = 1$ in Fig. 9. $d$ is the first to join and its *TreeJoin* message is forwarded to $c$ (Fig. 9a). Since this message is the first joining request $c$ has received, $c$ handles the message as in a bypass tree. That is, $c$ relays the message to $s$ without becoming a forwarder, and $s$ directly serves $d$. Later on, when $b$ joins, its *TreeJoin* message goes through $c$ to $s$ (Fig. 9b). Now $c$ has received two joining messages and this number exceeds its message threshold 1. Therefore, $c$ handles the message as in an embedded tree and turns itself into a forwarder. Afterwards, when $f$ joins, $s$ directly serves

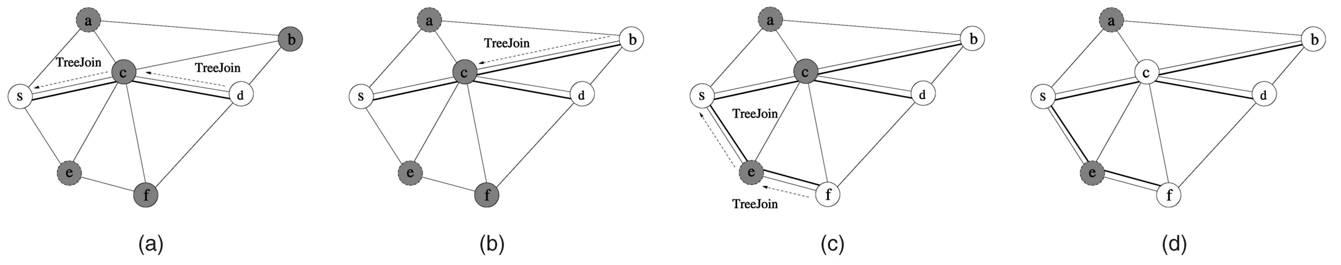


Fig. 7. An example of building an embedded tree. Tree branches are indicated by bold lines.

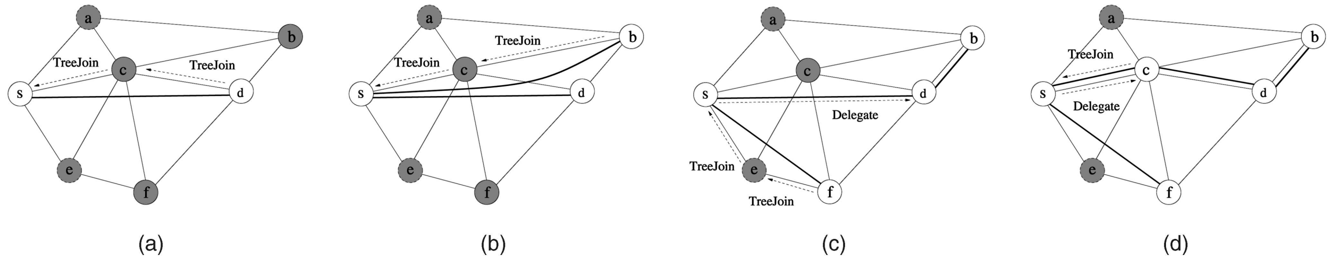


Fig. 8. An example of building a bypass tree. Tree branches are indicated by bold lines.

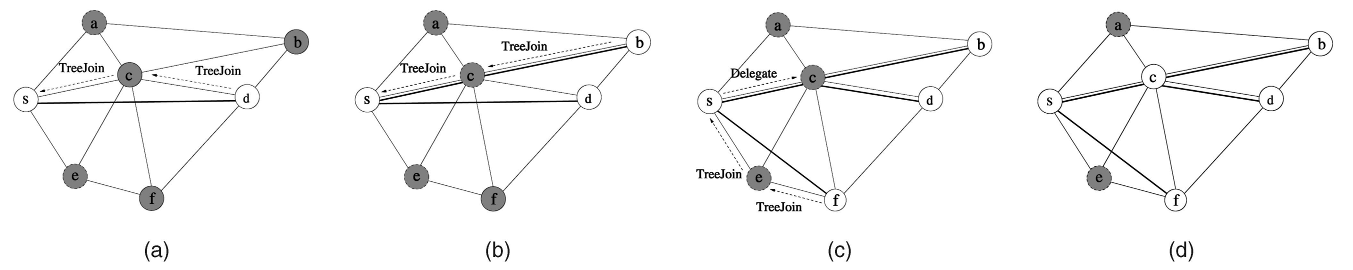


Fig. 9. An example of building an intermediate tree. Tree branches are indicated by bold lines.

$f$ by skipping $e$. Now $s$ has three children, i.e, $c, d$, and $f$. $s$ then delegates $d$ to $c$ as $\angle dsc$ is the minimum among all angles formed by $s$'s children (Fig. 9c). Finally, when $c$ joins the group, it does not need to send any *TreeJoin* message since it is already a forwarder (Fig. 9d).

## 5 Illustrative Numerical Results

### 5.1 Simulation Setup

We generate 10 *Transit-Stub* topologies with GT-ITM [18]. Each topology is a two-layer hierarchy of transit networks and stub networks. Following [10], each topology in our simulations has four transit domains (each with 16 randomly distributed routers on a $1{,}024 \times 1{,}024$ grid) and 64 stub domains (each with 15 randomly distributed routers on a $32 \times 32$ grid). A host is connected to a stub router via a LAN (of $4 \times 4$ grid). The delays of LAN links are 1 ms, and the delays of core links are given by the topology generator. For each group, we randomly select a host as the source to disseminate packets. For GNP, we select 20 landmarks based on the $N$-cluster-median criterion as in [11]. We use the following metrics to evaluate our scheme:

- *Relative delay penalty (RDP)*, defined as the ratio of the overlay delay from the source to a given host to the delay along the shortest unicast path between them [19].
- *Link stress*, defined as the number of copies of a packet transmitted over a certain physical link [19]. Similarly, we define *node stress* of a host as the number of the host's children in an overlay tree.
- *Normalized network resource usage*, defined as the summation of the delays of all overlay paths in an overlay tree divided by the summation of the delays of all underlay links in an IP-multicast tree [19].

Unless otherwise stated, the parameters we use are $N = 1{,}024$ (a total of 1,024 hosts in the session, or in the system), $G = 128$ (128 of them belong to the same group), $K = 8$ (the maximum node stress of a host is 8), $MessageThreshold \equiv R = 8$ (for an intermediate tree), and $T = 5$ degrees (the angle between two adjacent children should be larger than 5 degrees).

### 5.2 Performance of SMesh

We first compare SMesh's performance with a traditional overlay tree protocol, Narada [19]. Since Narada does not consider multiple groups, we set $G = N$ for a fair comparison. In this case, SMesh is similar to DT. We further compare meshes with GNP coordinates and with geographic coordinates. Fig. 10a shows the average RDP versus the session size. In general, RDP increases with the session size. DT with GNP performs the best, especially when the session size is large. This is because GNP coordinates accurately estimate host locations in the Internet. For a medium or large session (more than 64 hosts), DT with GNP achieves significantly lower RDP than Narada. Regarding link stress, DT with GNP performs better than Narada (Fig. 10b). It also performs better than DT with geographic coordinates for sparse network. For a dense network, using geographic coordinates may perform better. This is because the higher member density, the higher probability that DT with GNP has "small angles." As a result, overlay paths are likely to pass through the same underlay links, leading to high stresses. Fig. 10c shows the network resource usage. Since DT with GNP consists of low-delay paths, its resource usage is the lowest.

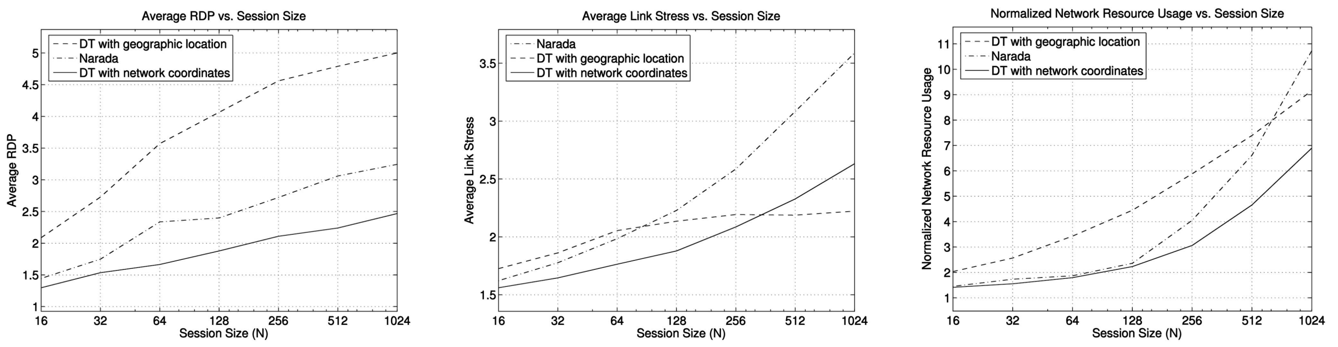
### 5.3 Performance of Embedded, Bypass, and Intermediate Trees

We compare SMesh trees (embedded, bypass, and intermediate trees) with Narada and Scribe in Fig. 11. Please refer to Section 6 or [30] for more details of Scribe. In the simulations, Scribe has the same group size as SMesh, and Narada builds an independent tree for each group.

Fig. 11a shows the average RDP versus the group size $G$. The RDP of a bypass tree is significantly lower than that of an embedded tree, while an intermediate tree lies between them. The RDP of an embedded tree is independent of group size as its tree edges are all mesh edges. For small groups, a bypass tree skips nearly all mesh edges; therefore, its RDP is close to one. When the group is large, a $TreeJoin$ message will meet a group member instead of a forwarder with high probability. A bypass tree is hence similar to an embedded tree. Therefore, as the group size increases, the RDP of a bypass tree approaches that of an embedded tree.

In the figure, Narada has higher RDP than intermediate and bypass trees, but lower RDP than an embedded tree. As mentioned, the embedded tree is not efficient for small groups. However, when the group size increases, its RDP does not increase much. On the other hand, Scribe suffers the highest RDP for almost all group sizes. One possible reason is that each Scribe node is assigned a random key for overlay routing. While the key of a node is uncorrelated to its network location, overlay routing based on keys is not efficient.

Fig. 11b compares link stresses of the trees. An embedded tree has the lowest stress, because its forwarding load is distributed to all the hosts in the network.



(a) (b) (c)

Fig. 10. Performance comparison of DT mesh using GNP, DT mesh using geographic coordinates, and Narada. (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

Bypass and intermediate trees have higher link stresses due to their higher node stresses. An intermediate tree has slightly higher stress than a bypass tree, mainly due to high stresses at some hosts. On the other hand, both Scribe and Narada suffer higher link stresses than SMesh trees. Scribe does not have degree bound for hosts. So, some hosts may have high node stresses, which further leads to high stresses for some links.

Fig. 11c shows the normalized resource usage of the trees. A bypass tree achieves lower resource usage than an intermediate tree, because it does not incur unnecessary detours to some intermediate nonmember hosts. As compared to an embedded tree, a bypass tree achieves lower resource usage for small groups, mainly due to fewer overlay hops from source to hosts. However, as the group size $G$ increases, a bypass tree has higher resource usage than an embedded tree. In this case, a bypass tree is less efficient than an embedded tree.

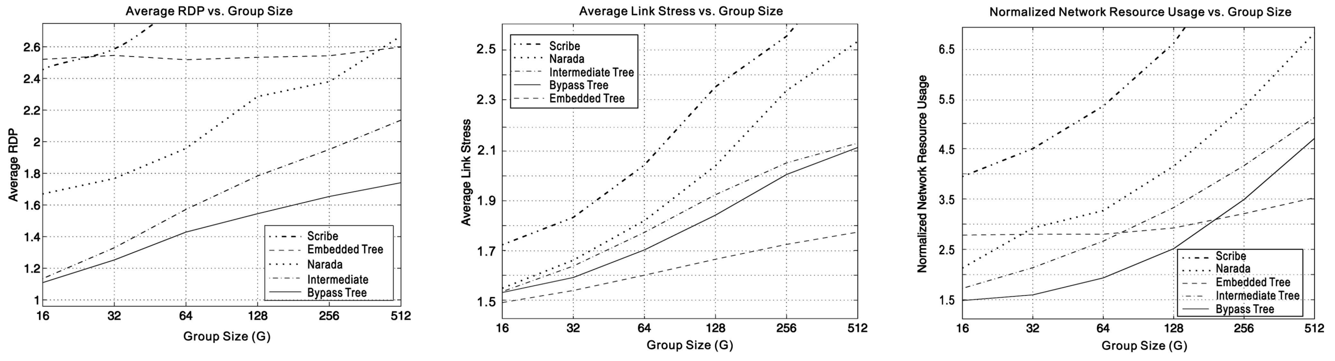
From the figure, Scribe has the highest normalized resource usage and Narada has the second highest. This is not surprising as Scribe has the highest RDP and stress, and Narada has the second highest stress. These results show that they are not efficient to connect small groups from a large pool of hosts. If we can build a stable mesh among all hosts and construct trees on top of it, the network resource for data delivery can be significantly reduced.

In summary, the results show that the bypass tree works well overall. It achieves low RDP and low resource usage, with intermediate stress performance. For an intermediate tree, its RDP and resource usage lie between bypass and embedded trees, especially for small to medium groups.

We show in Figs. 12a and 12b the accumulative distribution of link stress and node stress, respectively. An embedded tree has the lowest link stress. The proportion of links experiencing high stress is low (less than 10 percent links have stress higher than 3). Regarding node stress, there are around 50 percent hosts with zero node stress for both bypass and intermediate trees. These hosts are all leaf nodes in trees. For an embedded tree, because all hosts participate in the routing process, the percentage of hosts with zero node stress is much lower. In all cases, most hosts have low node stresses (less than 4). This shows that our schemes achieve good load balancing.

## 5.4 Sensitivity Analysis

We now examine the effect of system parameters on tree performance. We first investigate the impact of message threshold to intermediate tree. We plot in Fig. 13 the RDP, link stress, and network resource usage versus the message threshold. Note that the cases of threshold equal to 0 and $G$ ($G = 128$ in this case) correspond to an embedded tree and a bypass tree, respectively. The threshold values in-between correspond to intermediate trees. By adjusting



(a) (b) (c)

Fig. 11. Performance comparison of embedded, bypass, and intermediate trees ($N = 1{,}024, K = 8, R = 8$, and $T = 5$ degrees). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

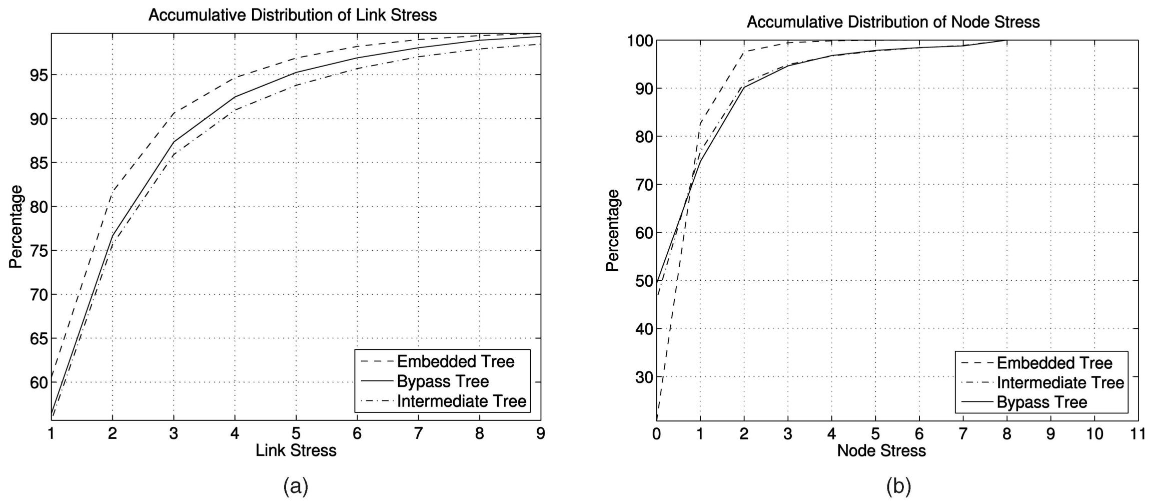


Fig. 12. Stress distribution in embedded, bypass, and intermediate trees ($N = 1{,}024, G = 128, K = 8,$ and $T = 5$ degrees). (a) Accumulative distribution of link stress and (b) accumulative distribution of node stress.

the threshold, the intermediate tree achieves performance somewhere between bypass and embedded trees. There is clearly a trade-off between RDP, stress, and resource usage. For example, when $R = 1$, the intermediate tree achieves higher (lower) RDP than a bypass tree (embedded tree). Meanwhile, it achieves lower (higher) link stress than a bypass tree (embedded tree).

We examine bypass and embedded trees in the following. Recall that the angle threshold, $T$, trades off end-to-end delay with network resource usage. We show the RDP, link stress, and network resource usage versus $T$ in Fig. 14. The average RDP of an embedded tree is higher than that of a bypass tree, and is quite independent of $T$. On the other hand, the RDP of a bypass tree increases with $T$ due to more delegations. The link stresses of the two trees both quickly decrease with $T$ at the beginning. This is because we have used path aggregation. Later on, the link stress converges to a stable value. The network resource usage shows similar trends to the link stress. The results show that a bypass tree is in general better than an embedded tree, and $T$ does not need to be high to achieve a good performance.

## 6 Related Work

In this section, we discuss related work on P2P streaming and overlay construction. In one-to-many multimedia streaming and communications applications, an effecient approach is to use IP multicasting [20]. Today, many of the existing networking infrastructure are multicast capable. Emerging commercial video transport and distribution networks heavily make use of IP multicasting. However, there are many operational issues that limit the use of IP multicasting into individual autonomous networks. Furthermore, only trusted hosts are allowed to be multicast sources. Thus, while it is highly efficient, IP multicasting is still not an option for P2P streaming at the user level.

As a comparison, in a P2P overlay network, hosts are responsible for packets replication and forwarding. A P2P network only uses unicast and does not need multicast-capable routers. It is, hence, more deployable and flexible. Currently, there are two types of overlays for P2P streaming: tree structure and gossip mesh. The first one builds one or multiple overlay tree(s) to distribute data among hosts. Examples include application-layer multicast protocols (e.g., Narada and NICE) and some P2P video-on-demand systems (e.g., P2Cast and P2VoD) [19], [21], [22], [23], [24], [25]. The

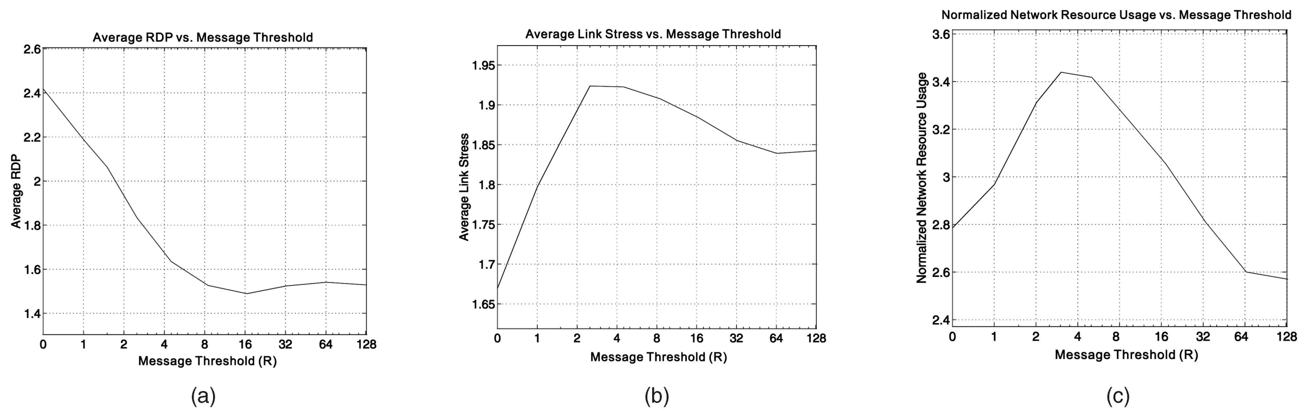


Fig. 13. Performance of an intermediate tree with different message thresholds $R$ ($N = 1{,}024, G = 128, K = 8,$ and $T = 5$ degrees). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

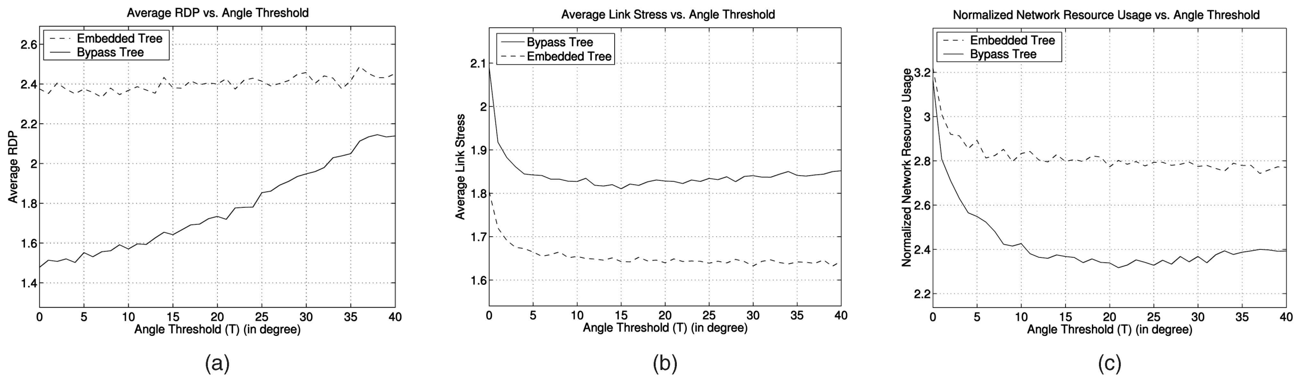


Fig. 14. Tree performance with different angle thresholds $T$ ($N = 1{,}024, G = 128,$ and $K = 8$). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

second one builds a mesh among hosts using gossip algorithms, with hosts exchanging data with their neighbors in the mesh [1], [3], [26]. The gossip-based approaches achieve high resilience to network and group dynamics. However, they have high control overhead due to data scheduling and mesh maintenance. They also have high playback delay because in the gossip mesh, a host may not always find close peers as their neighbors. On the contrary, trees introduce lower end-to-end delay and are easier to maintain.

Our work in this paper falls into the first category. Although we build a mesh spanning hosts, we do not directly use mesh edges for data exchange. Instead, we build overlay trees on top of the mesh for data exchange. In order to improve tree resilience, we may incorporate additional loss recovery schemes into our system [27], [28].

We now compare our work with other P2P tree construction protocols. Most previously proposed tree-based protocols build a single overlay tree rather than dealing with multiple dynamic trees as we investigate here [10], [19], [21], [29]. To support multiple groups or channels, these protocols have to build multiple independent trees. As a result, host joining or leaving of a group leads to reconstruction of the tree. Such cost may be high when hosts frequently change their groups. SMesh addresses this problem by using a relatively stable mesh consisting of all hosts, even though the membership of each group can be quite dynamic. SMesh is similar to Scribe [30], [31] and Bayeux [32], [33] in the sense that a shared overlay mesh is built before multiple independent delivery trees are built. SMesh differs from and improves them in two aspects: 1) Scribe and Bayeux trees are embedded in the mesh. We study the case where tree branches may bypass mesh edges. With packets taking fewer hops to reach their group members, a bypass tree is more efficient in terms of end-to-end delay and resource usage. 2) Each host in Scribe and Bayeux is assigned a random key (a kind of logical address), which is uncorrelated to its network location. The resultant mesh is often inefficient. Topology-aware mesh construction may improve routing efficiency, but it requires high probing overhead for topology inference [34], [35]. In contrast, SMesh builds an efficient DT mesh based on GNP coordinates. This mesh is distributed, adaptive to host dynamics and of low probing overhead. A preliminary version of SMesh has been studied in [36]. In this paper, we propose a distributed algorithm for partition detection and recovery in the mesh. We also study an aggregation and delegation algorithm to improve tree performance. Finally, we present more comprehensive simulation results.

## 7 Conclusion

In P2P streaming networks, users may frequently hop from one group to another. In this paper, we propose a novel framework called SMesh to serve dynamic groups for Internet streaming. SMesh supports multiple groups and can efficiently distribute data to these dynamic groups. It first builds a shared overlay mesh for all hosts in the system. The stable mesh is then used to guide the construction of data delivery trees for each group. We study three ways to construct a tree, i.e., embedded, bypass, and intermediate trees. We also propose and study an aggregation and delegation algorithm to balance the load among hosts, which trades off end-to-end delay with lower network resource usage.

Through simulations on Internet-like topologies, we show that SMesh achieves low RDP and low link stress as compared to traditional tree-based protocols. In our simulations, a bypass tree performs better than an embedded tree in terms of RDP but not so for link stress. By adjusting message threshold, an intermediate tree can achieve performance between bypass and embedded trees.

## Acknowledgments

This work was supported, in part, by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611107), and the Cisco University Research Program Fund, a corporate advised fund of Silicon Valley Community Foundation (SVCF08/09.EG01).

## References

[1] X. Zhang, J. Liu, B. Li, and T.-S.P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Peer-to-Peer Live Media Streaming," *Proc. IEEE INFOCOM '05,* pp. 2102-2111, Mar. 2005.
[2] X. Liao, H. Jin, Y. Liu, L.M. Ni, and D. Deng, "Anysee: Peer-to-Peer Live Streaming," *Proc. IEEE INFOCOM '06,* Apr. 2006.
[3] Y. Tang, J.-G. Luo, Q. Zhang, M. Zhang, and S.-Q. Yang, "Deploying P2P Networks for Large-Scale Live Video-Streaming Service," *IEEE Comm. Magazine,* vol. 45, no. 6, pp. 100-106, June 2007.
[4] PPLive, http://www.pplive.com, 2009.

[5] X. Hei, Y. Liu, and K.W. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," *IEEE J. Selected Areas in Comm.*, vol. 25, no. 9, pp. 1640-1654, Dec. 2007.

[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672-1687, Dec. 2007.

[7] M. Cha, P. Rodriguez, J. Crowcroft, S. Moon, and X. Amatriain, "Watching Television over an IP Network," *Proc. ACM Internet Measurement Conf. (IMC '08)*, pp. 71-83, Oct. 2008.

[8] Skype, http://www.skype.com/, 2009.

[9] D. Rossi, M. Mellia, and M. Meo, "A Detailed Measurement of Skype Network Traffic," *Proc. Int'l Workshop Peer-To-Peer Systems (IPTPS '08)*, Feb. 2008.

[10] J. Liebeherr, M. Nahas, and W. Si, "Application-Layer Multicasting with Delaunay Triangulation Overlays," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 8, pp. 1472-1488, Oct. 2002.

[11] T.S.E. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," *Proc. IEEE INFOCOM '02*, pp. 170-179, June 2002.

[12] L. Tang and M. Crovella, "Virtual Landmarks for the Internet," *Proc. ACM Internet Measurement Conf. (IMC '03)*, pp. 143-152, Oct. 2003.

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A Decentralized Network Coordinate System," *Proc. ACM SIGCOMM '04*, pp. 15-26, Aug. 2004.

[14] H. Lim, J.C. Hou, and C.-H. Choi, "Constructing Internet Coordinate System Based on Delay Measurement," *IEEE/ACM Trans. Networking*, vol. 13, no. 3, pp. 513-525, June 2005.

[15] E. Kranakis, H. Singh, and J. Urrutia, "Compass Routing on Geometric Networks," *Proc. Canadian Conf. Computational Geometry (CCCG '99)*, pp. 51-54, Aug. 1999.

[16] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2000.

[17] R. Sibson, "Locally Equiangular Triangulations," *Computer J.*, vol. 3, no. 21, pp. 243-245, 1978.

[18] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," *Proc. IEEE INFOCOM '96*, pp. 594-602, Mar. 1996.

[19] Y.H. Chu, S. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 8, pp. 1456-1471, Oct. 2002.

[20] S.E. Deering, "Multicast Routing in Internetworks and Extended LANs," *ACM SIGCOMM Computer Comm. Rev.*, vol. 18, no. 4, pp. 55-64, Aug. 1988.

[21] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," *Proc. ACM SIGCOMM '02*, pp. 205-217, Aug. 2002.

[22] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2Cast: Peer-to-Peer Patching Scheme for VoD Service," *Proc. Int'l World Wide Web Conf. (WWW '03)*, pp. 301-309, May 2003.

[23] T. Do, K.A. Hua, and M. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," *Proc. IEEE Int'l Conf. Comm. (ICC '04)*, pp. 1467-1472, June 2004.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," *Proc. ACM Symp. Operating Systems Principles (SOSP '03)*, pp. 298-313, Oct. 2003.

[25] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *Proc. ACM Symp. Operating Systems Principles (SOSP '03)*, pp. 282-297, Oct. 2003.

[26] H. Chi, Q. Zhang, J. Jia, and X. Shen, "Efficient Search and Scheduling in P2P-Based Media-on-Demand Streaming Service," *IEEE J. Selected Areas in Comm.*, vol. 25, no. 1, pp. 119-130, Jan. 2007.

[27] W.-P. Yiu, K.-F. Wong, S.-H. Chan, W.-C. Wong, Q. Zhang, W.-W. Zhu, and Y.-Q. Zhang, "Lateral Error Recovery for Media Streaming in Application-Level Multicast," *IEEE Trans. Multimedia*, vol. 8, no. 2, pp. 219-232, Apr. 2006.

[28] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan, "Resilient Multicast Using Overlays," *IEEE/ACM Trans. Networking*, vol. 14, no. 2, pp. 237-248, Apr. 2006.

[29] X. Jin, K.-L. Cheng, and S.-H.G. Chan, "SIM: Scalable Island Multicast for Peer-to-Peer Media Streaming," *Proc. IEEE Int'l Conf. Multimedia & Expo (ICME '06)*, pp. 913-916, July 2006.

[30] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE J. Selected Areas in Comm.*, vol. 20, no. 8, pp. 1489-1499, Oct. 2002.

[31] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329-350, Springer, Nov. 2001.

[32] S.Q. Zhuang, B.Y. Zhao, A.D. Joseph, R.H. Katz, and J.D. Kubiatowicz, "Bayeux: An Architecture for Scalable and Fault-Tolerant Wide Area Data Dissemination," *Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '01)*, June 2001.

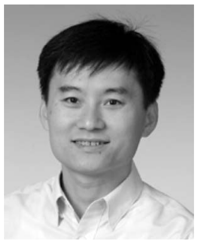
[33] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE J. Selected Areas in Comm.*, vol. 22, no. 1, pp. 41-53, Jan. 2004.

[34] M. Castro, M.B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays," *Proc. IEEE INFOCOM '03*, pp. 1510-1520, Apr. 2003.

[35] X. Jin, W.-P.K. Yiu, S.-H.G. Chan, and Y. Wang, "Network Topology Inference Based on End-to-End Measurements," *IEEE J. Selected Areas in Comm.*, vol. 24, no. 12, pp. 2182-2195, Dec. 2006.

[36] X. Jin, W.-C. Wong, and S.-H.G. Chan, "Serving Dynamic Groups in Application-Level Multicast," *Proc. Int'l Workshop High Performance Switching and Routing (HPSR '05)*, May 2005.

**Xing Jin** received the BEng degree in computer science and technology from Tsinghua University, Beijing, China, in 2002, and the PhD degree in computer science and engineering from The Hong Kong University of Science and Technology (HKUST), Kowloon, in 2007. He is currently a member of Technical Staff in the Systems Technology Group at Oracle, Redwood Shores, California. His research interests include distributed information storage and retrieval, peer-to-peer technologies, multimedia networking, and Internet topology inference. He is a member of Sigma Xi and IEEE COMSOC Multimedia Communications Technical Committee. He has been on the editorial board of *Journal of Multimedia* since 2006, and *Canadian Journal of Pure and Applied Sciences* since 2007. He was awarded the Microsoft Research Fellowship in 2005.



**S.-H. Gary Chan** received the BSE degree (highest honor) in electrical engineering from Princeton University, New Jersey, in 1993, with certificates in applied and computational mathematics, engineering physics, and engineering and management systems, and the MSE and PhD degrees in electrical engineering from Stanford University, California, in 1994 and 1999, respectively, with a minor in business administration. He is currently an associate professor with the Department of Computer Science and Engineering, HKUST, Kowloon, and an adjunct researcher with Microsoft Research Asia, Beijing. He was a visiting assistant professor in Networking with the Department of Computer Science, University of California, Davis, from 1998 to 1999. His research interests include multimedia networking, peer-to-peer technologies and streaming, and wireless communication networks. He is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa. He was a William and Leila fellow at Stanford University during 1993-1994. At Princeton University, he was the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence in 1993. He served as a vice-chair of IEEE COMSOC Multimedia Communications Technical Committee from 2003 to 2006. He is a guest editor for the *IEEE Communication Magazine* (Special Issues on Peer-to-Peer Multimedia Streaming), 2007 and Springer *Multimedia Tools and Applications* (special issue on advances in consumer communications and networking), 2007. He is the cochair of the Multimedia Symposium for IEEE ICC (2007). He was the cochair for the workshop on Advances in Peer-to-Peer Multimedia Streaming for the ACM Multimedia Conference (2005), and the Multimedia Symposia for IEEE GLOBECOM (2006) and IEEE ICC (2005). He is a senior member of IEEE Computer Society.

**Wan-Ching Wong** received the BSc and MPhil degrees in computer science from HKUST, Kowloon, in 2001 and 2003, respectively. He was a research assistant at HKUST until August 2003. His research interests include Internet technologies, peer-to-peer networks, and aspect-oriented programming.

**Ali C. Begen** is a software engineer in the Video and Content Platforms Research and Advanced Development Group at Cisco, San Jose, California, where he participates in video transport and distribution projects. His interests include networked entertainment, multimedia transport protocols, and content distribution. He has the PhD degree in electrical and computer engineering from the Georgia Institute of Technology. He received the Best Student-Paper Award at IEEE ICIP 2003. He received the Most-Cited Paper Award from *Elsevier Signal Processing: Image Communication,* in 2008. He is a member of the IEEE Computer Society and the ACM.