International Journal of Research in Computer Science
ISSN 2249-8257 Volume 1 Issue 1 (2011) pp. 9-24

www.ijorcs.org

Efficiently Delivering Data Packets Using Distributed Protocol for Runtime Groups Formed In Peer-to-Peer Network

Yasa Ramya[1] , K. Bhagyalaxmi[2]

[1]*Department of C. S. E. TKR College of Engineering & Technology, Meerpet, Hyderabad, India*
Email: yasa_ramya@yahoo.co.in

[2]*Department of C. S. E. TKR College of Engineering & Technology, Meerpet, Hyderabad, India*
Email: bhagyareddy2001@yahoo.co.in

Abstract

Peer-to-Peer streaming has been widely used over the internet, where 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 (subset-mesh) for dynamic groups where users may frequently hop from one group to another. SMesh first builds a relatively stable mesh consisting of all hosts for control messaging. The mesh supports dynamic host joining and leaving, and construction of delivery trees. Using the Delaunay Triangulation protocol as an example, we show how to construct an efficient mesh with low maintenance cost. We also study about various tree constructions based on the mesh, including embedded, bypass, and intermediate trees.

Keywords: Data packets, distributed protocol, subset-mesh, GNP estimation, delaunay triangulation, euclidian space, partition detection, data delivery trees.

I. Introduction

The penetration of broadband Internet access, there has been an increasing interest in media streaming services. Recently, P2Pstreaming 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 uni-cast connections. They cache and relay data for each other, thereby eliminating the need for resourceful servers from the system. In a P2P streaming system, the server 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. 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. Peers are divided into multiple small groups, each corresponding to a channel. Peers in one group share and relay the same streaming content for each other. In fact, there are many other similar applications over the Internet. In the application, the system contains multiple groups with different sources and contents. 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. A more typical example is group chat of “Skype”. Skype allows up to around 100 users to chat together. In above applications, as peers may dynamically hop from one group to another 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 the unpopular. 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.

Here 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 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. 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.

II. Review On GNP And DT

A. Review on GNP Estimation

GNP estimates host co-ordinates in a multidimensional Euclidean space such that the distance between hosts correlates well with the measured round-trip time between them. 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:Jlandmark(L1;L2; . . . ;L_M)= $\sum^{n}_{Li,Lj€\{L1,..LM\}}$ (Li-Lj- RTT(i, j))^2, where M is the number of landmarks, Li and Lj 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, Jlandmark 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 {L1; L1; . . . ; L_M to minimize Jlandmark, any one set can be used.

B. Review on Delaunay Triangulation

In the traditional DT protocol, each host knows its geographic coordinates. 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. In this approach, a host only needs to know the states of its immediate neighbours to construct and maintain the mesh, and the mesh is adaptive to dynamic host joining or leaving.

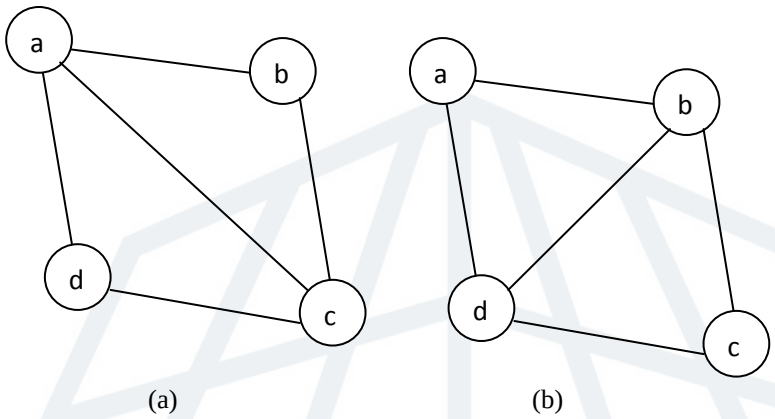


Fig.1. (a) Two adjacent triangles in a convex quadrilateral (4abc and 4adc) violate the DT property and (b) restore the DT property by disconnecting a, from c and connecting b and d.

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. 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 4abc and 4acd is smaller than that of 4abd and 4bcd. DT protocol then transforms the former configuration into the latter one. To achieve this, a host periodically sends HelloNeighbor messages to its neighbours to exchange their neighbourhood information. It removes a host from its neighbour list if the connection to that host violates the DT property. Similarly, a host adds another host into its neighbour 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(NlogN) messages.

Compass routing works as follows. Host a forwards messages with destination t to b, if b, among all a's neighbours, forms the smallest angle to t at a. We show an example in Fig. 2. Hosts b, c, and d are neighbours of host a, and a needs to forward a message to destination t. Since angle bat is the smallest among 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.

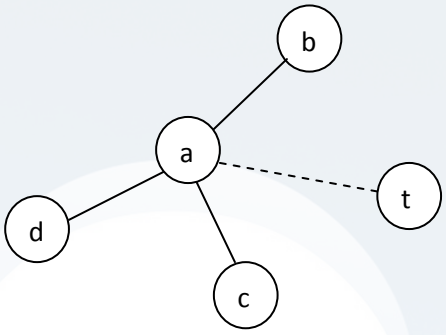


Fig.2. An example of compass routing. The angle bat is smaller than angles cat and dat. Therefore, when a receives a message with destination t, it forwards the message to b.

III. Mesh Formation And Maintenance

A. 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 □abc as the c
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 □abc as the s
□abc, which is certainly between 0 degree and 180 degrees. We show the examples of □+; □, and □ in Figs. 3a, 3b, and 3c, respectively.

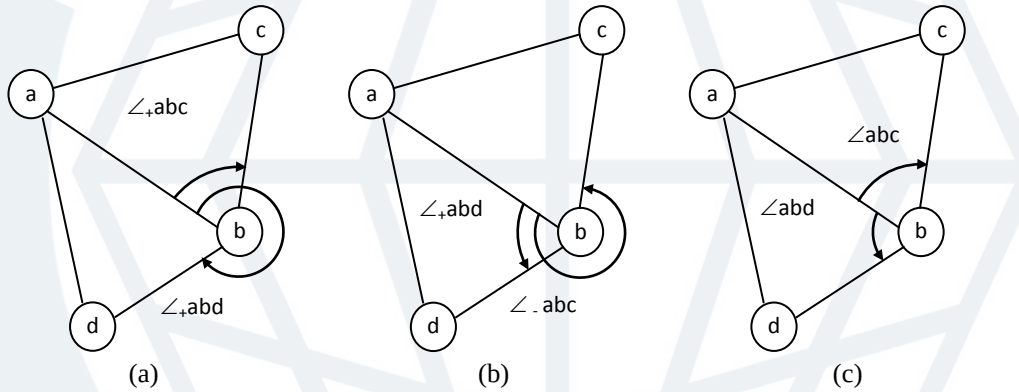


Fig. 3. Examples of (a) clockwise angles ∟+, (b) counter clockwise angles ∟-, and (c) undirected angles ∟.

We further consider two connected hosts b and c, and another host a in the graph. We say that c is the clockwise neighbour of b with respect to a if and only if □abc is less than 180 degrees and is the minimum among all the neighbours of b. In this case, we write $N^{+}_{b,a}$=c . For example, in Fig. 3a, c is the clockwise neighbour of b with respect to a(i.e., $N^{+}_{b,a}$=c).
Similarly, we say that c is the counter clockwise neighbour of b with respect to a, denoted as $N^{-}_{b,a}$=c, if and only if □abc is less than 180 degrees among all the neighbours of b. In Fig. 3b, d is the counter clock-wise neighbour of b with respect to a (i.e., $N^{-}_{b,a}$=d). Note that host b may not have any clockwise neighbour with respect to a.
For example, in Fig. 3, host b does not have any clockwise neighbour with respect to c, since angles □+cbd and □+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. $N^{+}_{u,t}$=Null or
2. $N^{-}_{u,t}$=Null or
3. □. $N^{-}_{u,t}$u. $N^{+}_{u,t}$>180
4. □$N^{+}_{u,t}$t. $N^{-}_{u,t}$>180

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

i) It 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 neighbours 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 . ∠$N^{-}_{u,t}$u. $N^{+}_{u,t}$ is as indicated.

ii) It 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 Δubc (as shown in Fig. 4d). When a host u receives a message with destination t, if it finds that. ∠$N^{+}_{u,t}$t. $N^{-}_{u,t}$>180 degrees and there is no connection.

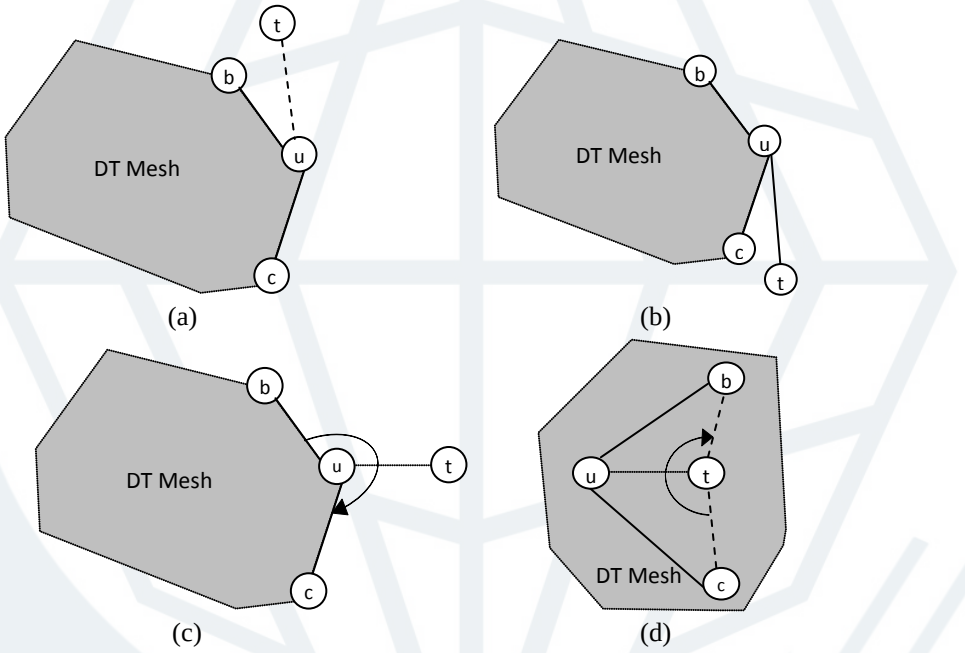


Fig. 4. (a), (b), and (c) Host u is on the boundary of the overlay mesh, and host t lies outside the mesh. (d) u is an interior host of the mesh. Host t lies inside triangle Δubc, but does not belong to the mesh.

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.

B. 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):

i) u first retrieves the list of landmarks by querying a host b with a GetLandmark message.Then u measures the round-trip time to the landmarks and estimates its coordinates.

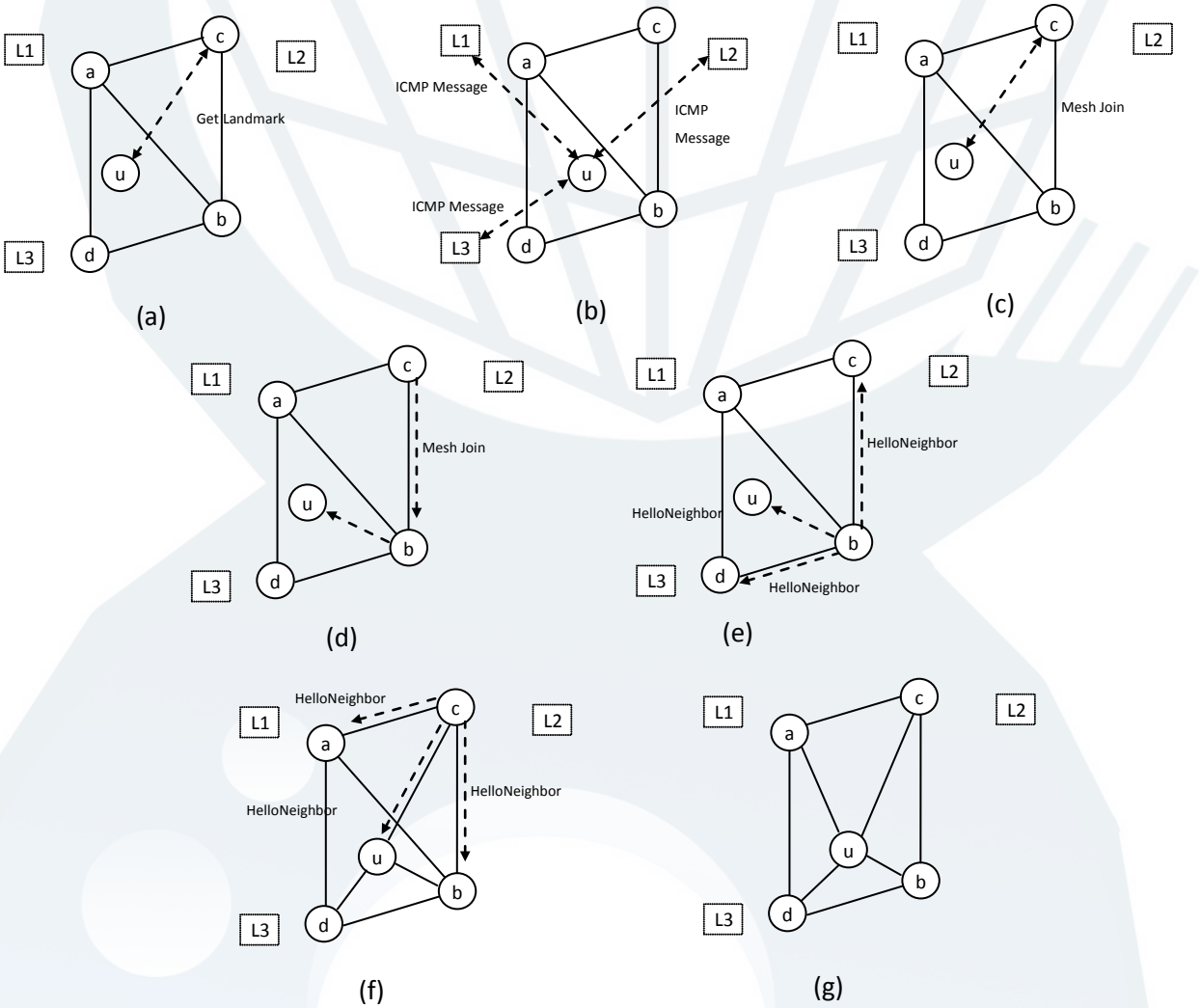
ii) After that, u sends a MeshJoin message to b.

iii) The message is then forwarded from b to c based on compass routing.

iv) Since u falls into Δacd; c knows that u is in another partitioned mesh. c then adds u into its neighbour listN_c to recover the partition. Note that the minimum internal angle of Δauc and Δabc is less than that of Δbuc and Δabu. Therefore, the connection between c and a violates the DT property, and c will remove a from N_c and notify a to remove the connection. c then broadcasts its neighbourhood information to its neighbours through HelloNeighbor messages. (In DT, each host needs to periodically send HelloNeighbor messages to its neighbours to exchange the neighbourhood information.)

v) Upon receiving HelloNeighbor messages from c, b, and d discover u. They add u into their neighbour lists since such connections do not violate the DT property. In the meantime, u also discovers b and d and adds them into its neighbour list. Suppose that b is the next to broadcast HelloNeighbor messages. Upon receiving the message, a discovers u and adds u into its neighbour list.

vi) The resultant overlay mesh after the joining of u still satisfies the DT property.



IV. Construction Of Data Delivery Trees

A. 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, non-member hosts may be included. 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 lie between embedded and bypass trees. In the following, we call a non leaf 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:

i) *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.

ii) *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 non-member 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.

iii) *Intermediate Tree*: We observe that an embedded tree requires the participation of non-member 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 non-member 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)

i) Me.Child[TreeJoin.InterestGroup]← Me.Child[TreeJoin.InterestGroup] U TreeJoin.JoinHost
ii) if TreeJoin.InterestGroup € Me.InterestGroups
iii) then Me.InterestGroups ←Me.InterestGroups U TreeJoin.InterestGroup
iv) TreeJoin.JoinHost ← Me
v) CompassRoute(TreeJoin,TreeJoin.GroupSource);

Algorithm 2

TREEJOINHANDLER_BYPASSTREE (TreeJoin)

i) if TreeJoin.InterestGroup € Me.InterestGroups
ii) then Me.Child[TreeJoin.InterestGroup] ←Me.Child[TreeJoin.InterestGroup] U TreeJoin.JoinHost
iii) else CompassRoute(TreeJoin, TreeJoin.GroupSource);

Algorithm 3

TREEJOINHANDLER_INTERMEDIATETREE (TreeJoin)

i) ReceivedMessage←ReceivedMessage+1
ii) if Received Message <=Message Threshold
iii) then TreeJoinHandler_BypassTree (TreeJoin);
iv) 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. 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.

B. Path Aggregation for QoS Provisioning

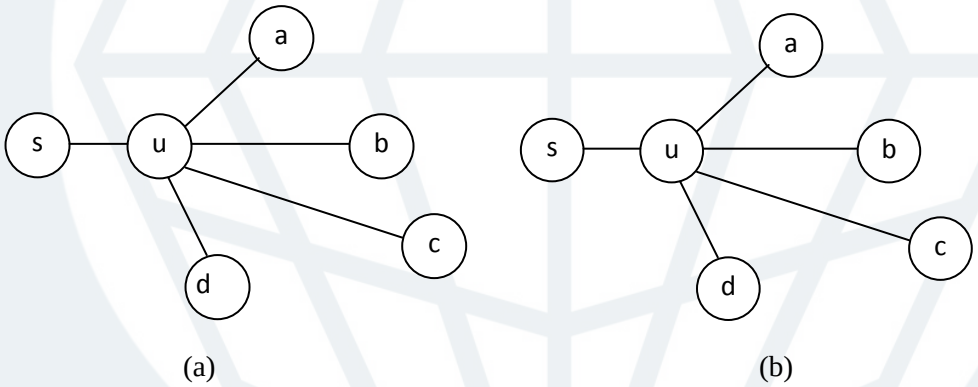
We note that the traditional DT protocol may result in high network resource consumption. For example, if host a belongs to domain A, and hosts b and b^1 belong to domain B, usually the delays of inter-domain paths ab and ab^1 are much higher than that of intra-domain path bb^1. In other words, angle $\angle bab^1$ is small. As a result, using compass routing, if either b or b^1 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, then 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 ∠buc is the sr
is smaller than the threshold T and because ||s – b|| < ||s –c||, u delegates c to b.

Algorithm 4

PATHAGGREGATION(u)

i) [c ,c^1]←a pair of children with the minimum adjacent angle
ii) while ∠c^1uc < T OR number of children > K
iii) do if ||c – s|| < ||c^1-s||
iv) then delegate c^1 to c
v) else elegate c to c^1
vi) [c, c^1] ← a pair of children with the minimum adjacent angle.



(a) (b)

Fig. 6. Host u delegates its child c to child b, since ∠buc < T and ||s –b|| < ||s –c||.

C. Illustrative Examples

We show in Figs. 7, 8, and 9 how embedded, bypass, and intermediate trees are constructed. White circles in figures, denote the 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 non-member, 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 non-member, 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 MessageT hreshold = 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 f by skipping e. Now s has three children, i.e, c; d, and f.s then delegates d to c as [illegible] (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).

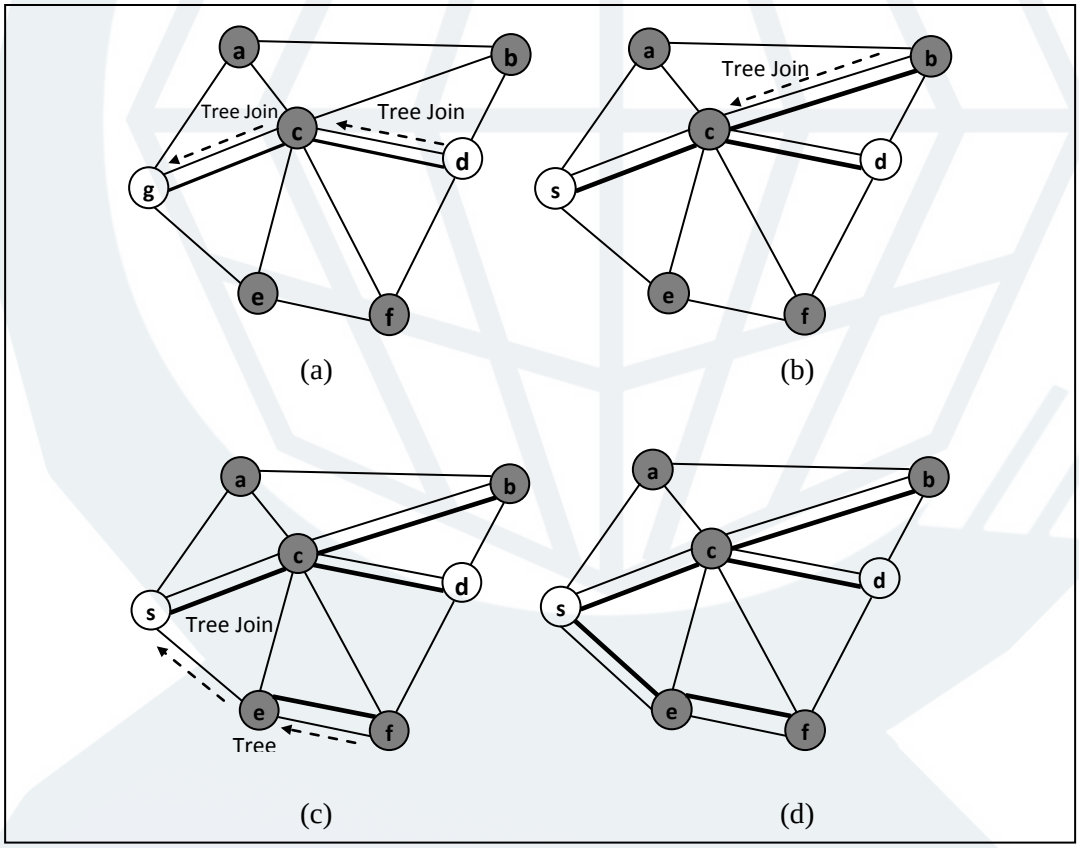


Fig.7 An example of building an embedded tree. Tree branches are indicated by bold lines.

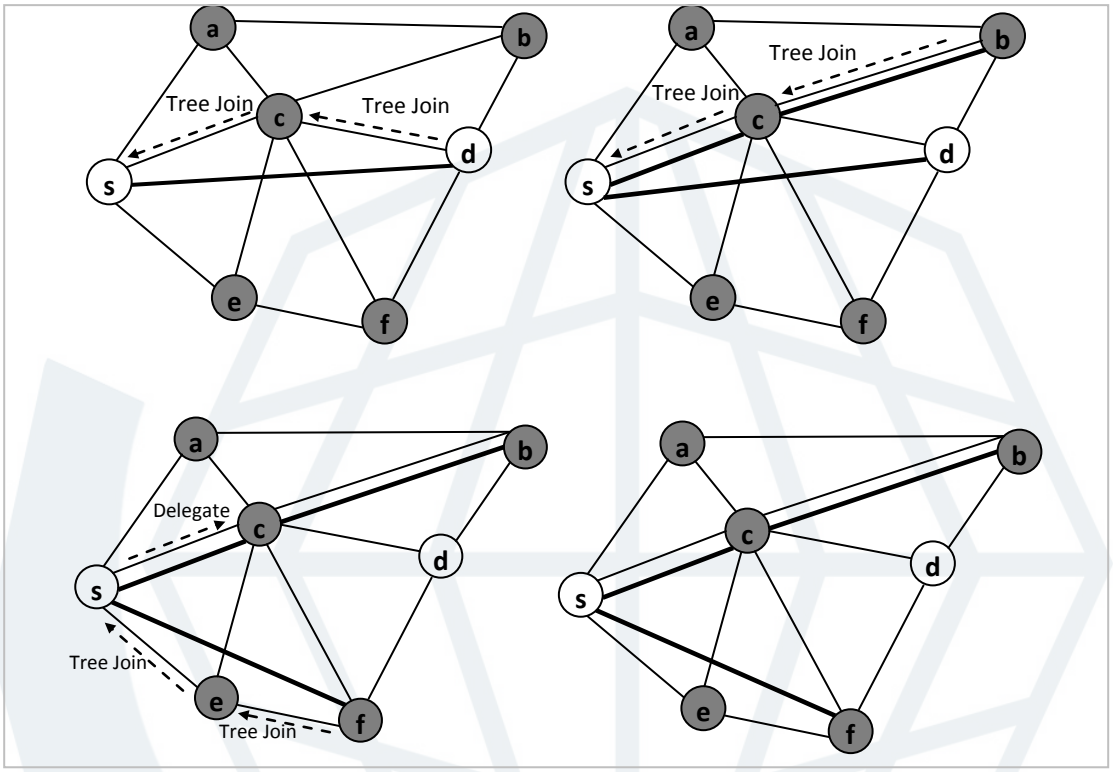


Fig. 8. An example of building a bypass tree. Tree branches are indicated by bold lines.

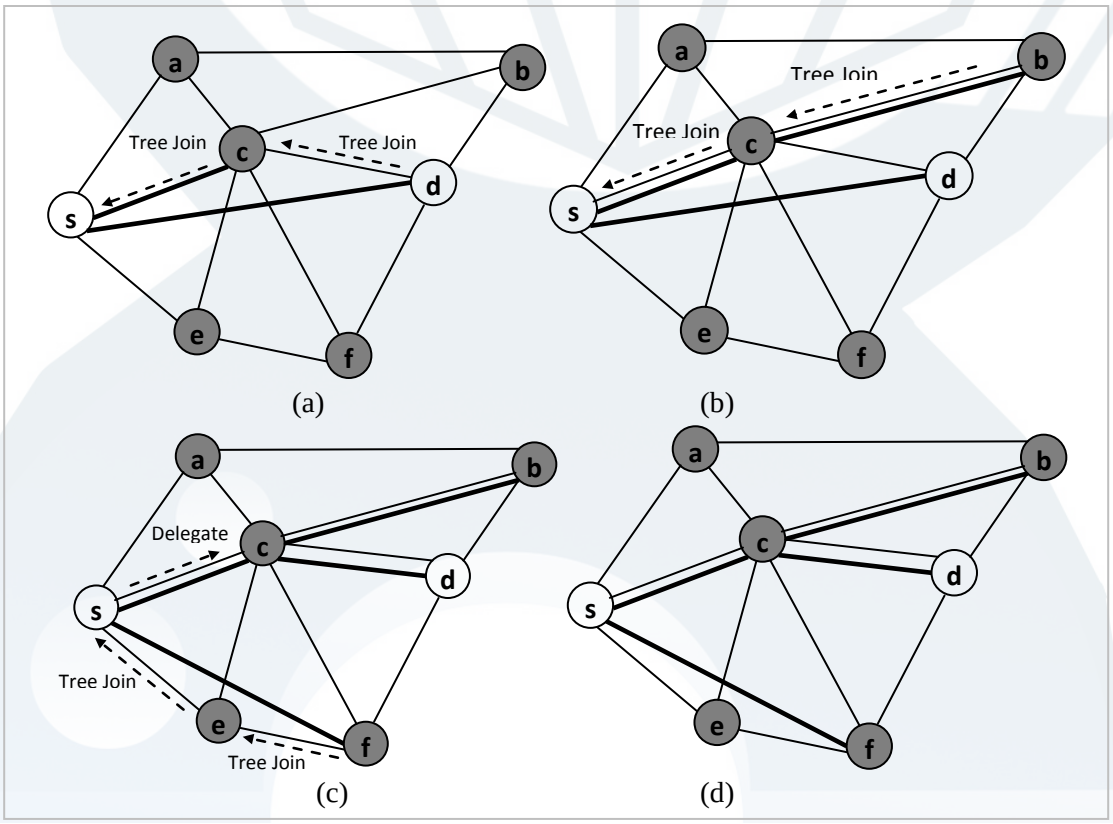


Fig. 9. An example of building an intermediate tree. Tree branches are indicated by bold lines

V. ILLUSTRATIVE NUMERICAL RESULTS

A. Simulation Setup

We generate 10 Transit-Stub topologies with GT-ITM. Each topology is a two-layer hierarchy of transit networks and stub networks. Following each topology in our simulations has four transit domains and 64 stub domains. A host is connected to a stub router via a LAN. 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. We use the following metrics to evaluate our scheme:

i) *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 uni-cast path between them.

ii) *Link stress*, defined as the number of copies of a packet transmitted over a certain physical link. Similarly, we define node stress of a host as the number of the host's children in an overlay tree.

iii) *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

B. Performance of SMesh

We first compare SMesh's performance with a traditional overlay tree protocol, Narada. 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, 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.

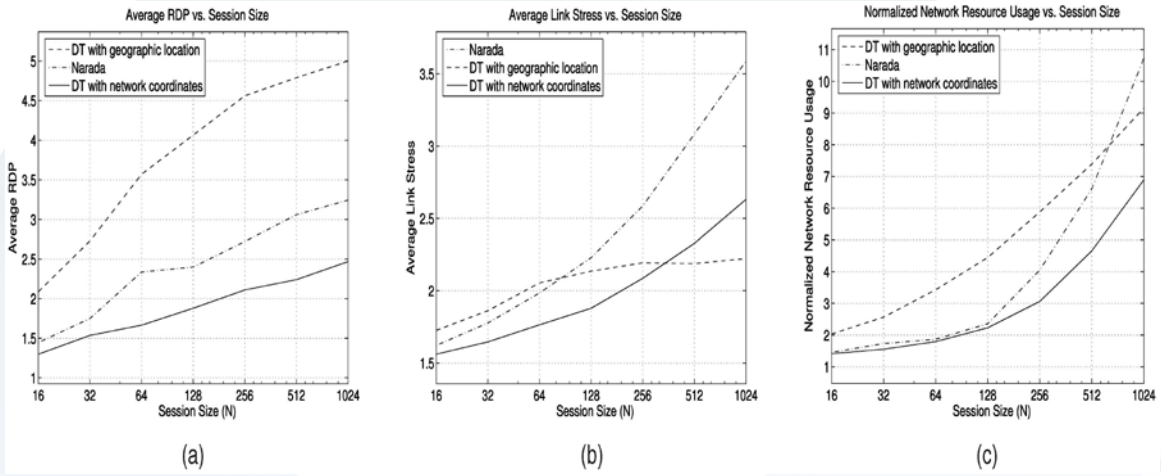


Fig. 10. Performance comparison of DT mesh using GNP, DT mesh using geographic coordinates, and Narada. (a) Average RDP, (b) average link stress, and (c) average normalized resource usage

C. Performance of Embedded, Bypass, and Intermediate Trees

We compare SMesh with Narada and Scribe in Fig. 11. Please refer to 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.

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. 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 non-member 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

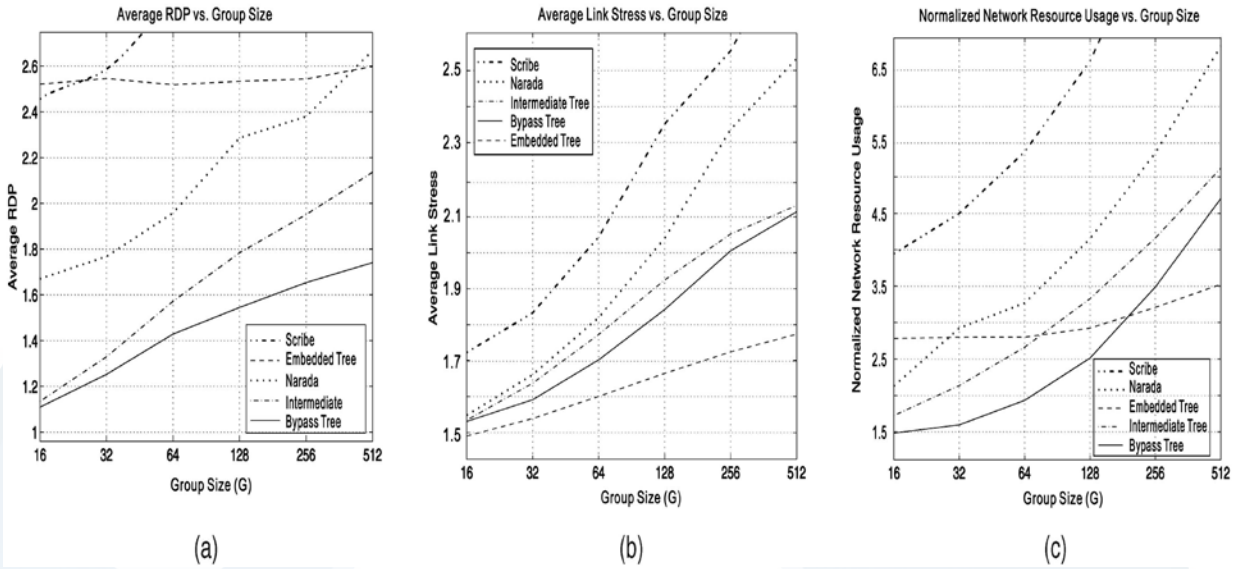


Fig. 11. Performance comparison of embedded, bypass, and intermediate trees (N=1024;K =8; R = 8, and T = 5 degrees). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

D. 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 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.

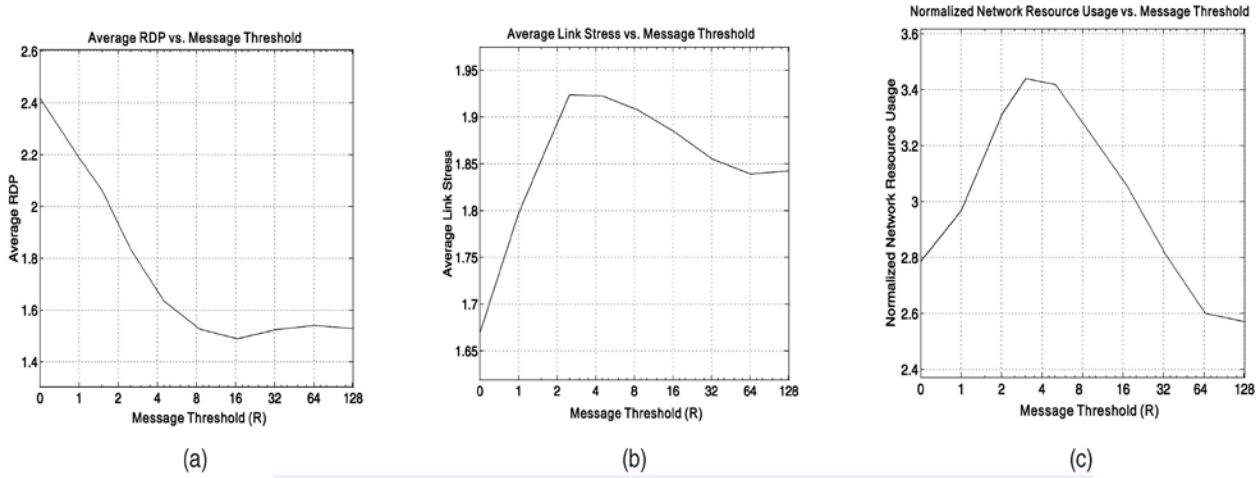


Fig. 13. Performance of an intermediate tree with different message thresholds R (N =1024;G = 128;K = 8, and T = 5 degrees). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage

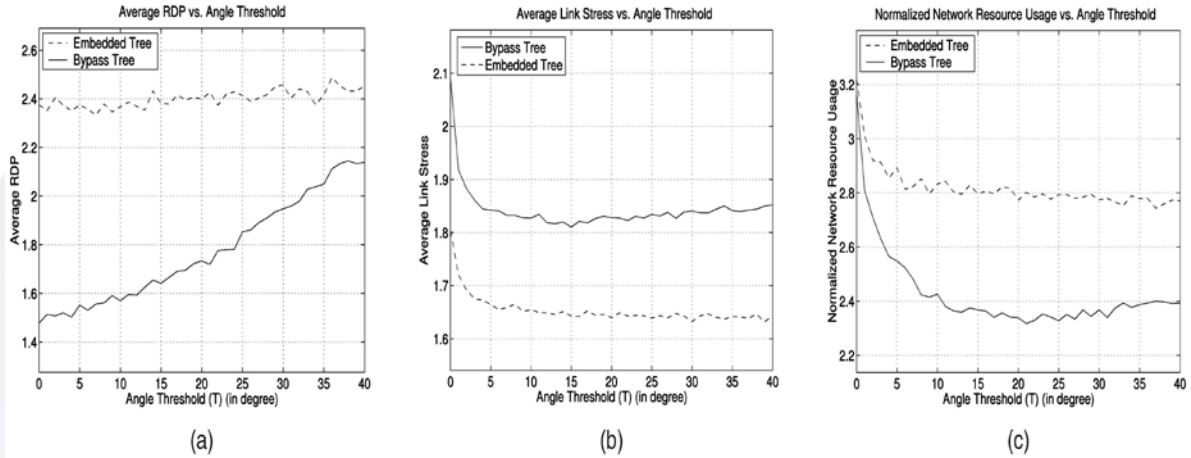


Fig. 14. Tree performance with different angle thresholds T (N = 1024;G =128, and K = 8). (a) Average RDP, (b) average link stress, and (c) average normalized resource usage.

VI. 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 efficient approach is to use IP multicasting. 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.

VII. 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.

VIII. 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).

IX. 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. doi:10.1109/INFOCOM.2006.288

[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. doi:10.1109/MCOM.2007.374426

[4] PPLive, http://www.pplive.com, 2009. 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.

[5] 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.

[6] 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.

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

[8] 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. doi:10.1109/JSAC.2002.803067

[9] T.S.E. Ng and H. Zhang, “Predicting Internet Network Distance with Coordinates-Based Approaches,” Proc. IEEE INFOCOM ’02, pp. 170-179, June 2002. doi:10.1109/INFCOM.2002.1019258

[10] L. Tang and M. Crovella, “Virtual Landmarks for the Internet,” Proc. ACM Internet Measurement Conf. (IMC ’03), pp. 143-152, Oct. 2003. doi:10.1145/948221.948223

[11] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A Decentralized Network Coordinate System,” Proc. ACM SIGCOMM ’04, pp. 15-26, Aug. 2004. doi:10.1145/1030194.1015471

How to cite

Yasa Ramya, Bhagyalaxmi, "Efficiently Delivering Data Packets Using Distributed Protocol for Runtime Groups Formed In Peer-to-Peer Network". *International Journal of Research in Computer Science*, 1 (1): pp. 9-24, September 2011. doi:10.7815/ijorcs.11.2011.002