MeshTree: Reliable Low Delay Degree-bounded Multicast Overlays

Su-Wei Tan, Gill Waters, John Crawford
Computing Laboratory, University of Kent, Kent, CT2 7NF, United Kingdom
Email: {swt3,A.G.Waters,J.S.Crawford}@kent.ac.uk

Abstract

We study decentralised low delay degree-constrained overlay multicast tree construction for single source real-time applications. This optimisation problem is NP-hard even if computed centrally. We identify two problems in traditional distributed solutions, namely the greedy problem and delay-cost trade-off. By offering solutions to these problems, we propose a new self-organising distributed tree building protocol called MeshTree. The main idea is to embed the delivery tree in a degree-bounded mesh containing many low cost links. Our simulation results show that MeshTree is comparable to the centralised Compact Tree algorithm, and always outperforms existing distributed solutions in delay optimisation. In addition, it generally yields trees with lower cost and traffic redundancy.

1. Introduction

This paper considers the problem of constructing "good" distribution trees for real-time applications such as audio/video conferencing and live webcasting from a data source. For these applications, low-latency delivery is of paramount importance. To accommodate large member populations, a cost-effective delivery mechanism such as multicasting is necessary. Since IP multicast has not been widely available, we consider the problem in the context of application layer multicast (ALM).

ALM implements multicast functions such as membership management and packet replication directly at the end systems. The end systems are organised into a logical overlay network, and multicast data using the overlay edges which are unicast tunnels. Hence ALM bypasses the need for network layer multicast support.

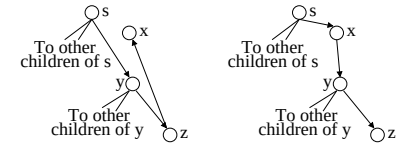
Creating an efficient overlay multicast tree is a challenging task. First, routing on top of the overlay results in redundant data traffic and prolonged end-to-end delay [8]. Secondly, end systems lack knowledge of the underlying topology, which is the key to building efficient overlays. As end systems often have limited processing power and available bandwidth, a degree constraint must be enforced in the delivery structure. In addition, the overlay structure is highly dynamic as it is formed by end systems that are prone to failures and may join/leave the session at will.

The above challenges and the requirement of low delay delivery in the real-time applications considered suggest that a solution must exhibit the following features.

- *Scalability*. This requires a decentralised scheme which imposes little protocol overhead (in terms of messages used to infer the node-to-node distances and to exchange the state information between the nodes).
- *Optimised Structure*. The overlay tree must be degree-bounded while providing low latency from source to receivers. This however, is an NP-hard optimisation problem [19].
- *Adaptability*. The solution must be able to react quickly to changes in the overlay membership (join/leave/failure) and network conditions.

We propose MeshTree, which fulfils the above properties in the following manner. First, MeshTree constructs overlay trees in a self-organising and fully distributed manner. The process uses limited network measurements and limited coordination between the nodes, hence has low overhead. MeshTree follows a mesh-based approach where the degree-bounded delivery tree is derived from a mesh. The mesh structure is inspired by the greedy problem and delay-cost trade-off (to be explained shortly) observed in distributed delay optimisation. We believe the structure offers a solution to the problems, and hence has low latency property. A mesh is also adaptable and provides good robustness.

We show by simulation experiments, MeshTree is comparable to the centralised Compact Tree algorithm [19]. In addition, it always performs better than a distributed scheme proposed by Banerjee et al. [5] — which we have previously shown [20] to outperform



(a) Sample Configuration (b) A Better Configuration

Figure 1. The greedy problem

switch-trees [11], HostCast [13], TBCP [14], HMTP [25] and AOM [23] in delay optimisation. MeshTree also offers quick failure recovery and is very responsive to changes in group membership.

Problems in Distributed Delay Minimisation: Intuitively, a good solution to the delay optimisation problem can be achieved with a delay-centric approach where nodes are placed as close as possible to the root. However, in a distributed environment where the degree-bounded nodes need to make decisions based on limited coordination using little knowledge about the topology, this approach can result in a greedy problem.

We explain this problem by using two existing decentralised protocols, i.e. switch-trees and HostCast. In these protocols, every node (except the root) maintains the delay from the root via the overlay tree to itself. Periodically, a node will try to improve its on-tree position by finding a better parent, i.e. a non-descendant node that provides a lower delay to the root. However, when a node has reached its degree bound, it will reject any new request from potential children. This can force nodes that are ideally placed near to the root to be placed far from the root. For example, Fig. 1(a) shows node x which is topologically close to the root, s, is positioned under z that is far from the root. As y (as well as other children of s) has found the best possible parent (the root), it will greedily stick to s. This may prevent a better configuration (e.g. see Fig. 1(b)) from happening.

The configuration in Fig. 1(b) suggests that the greedy problem can be avoided if the end systems are connected based on their relative position in the underlying topology, i.e. if nodes close by are connected together. As the aim is to construct a tree, we can view this as the minimum spanning tree problem with the delay between two nodes as the cost function. First, creating a degree-bounded minimum spanning tree is an NP-hard problem [12]. In addition, a low cost tree often results in high end-to-end delay [20]. Our proposed mesh structure addresses the above two problems.

In the next section, we position MeshTree with several related works. Section 3 describes MeshTree along with some evaluation results. Section 4 concludes the paper.

2. Related Work

In general, ALM protocols can be classified as either tree-based or mesh-based [9]. Several distributed tree-based protocols also attempt to create optimised delivery trees. Most notably, HMTP constructs low cost trees; HostCast, on the other hand, creates low delay trees; AOM attempts to achieve a balance between cost and delay; TBCP and switch-trees define generic techniques which can be adapted to different metrics, e.g. cost and delay. Our previous study has shown that the Banerjee at al. scheme that we use to compare with MeshTree, has superior delay performance to the above protocols. Yoid [10], another tree-based protocol, includes additional links to improve the tree robustness. However, these additional links are added without considering the degree constraint — hence, may not be useful in degree-bounded tree restoration. In addition, Yoid does not focus on overlay optimisation. NICE [4] and ZIGZAG [22] use a hierarchical cluster-based approach to construct overlay trees for large-scale applications. However, the resultant overlays are not degree-bounded based on a individual node's capacity constraint.

Several projects also consider the mesh-based approach for overlay construction. Narada [8] and Gossamer [7] run the path-vector protocol over a mesh overlay to derive a source-specific tree for each node, which is more suitable for many-to-many applications. Scribe [6] and CAN-multicast [16] build trees on top of the mesh overlay built with the distributed-hash table (DHT) techniques (i.e. Pastry [18] and CAN [15] respectively). However the DHT-based overlays provide scalable and robust data distribution at the expense of added difficulties in achieving a tree-wide optimisation [17]. The HyperCast project [3] studies the use of overlays based on geometric properties of logical graphs, e.g. hypercube and Delaunay triangles. The performance of these overlays depends highly on the mapping between the underlying network metrics and the geometric space.

3. MeshTree

MeshTree addresses the greedy problem and the delay-cost trade-off based on two simple ideas:

1. To solve the greedy problem, the overlay structure must contain a low cost tree which connects nodes that are topologically close together. The tree is called the backbone tree, and is rooted at the source node, s.

2. To improve the delay property of the backbone tree, shortcut links are added on top of the tree.

Essentially, this results in a mesh overlay. To fulfil the degree constraints, the mesh is degree-bounded based on each individual node's capacity limitation. The low delay tree can then be obtained from the mesh as the shortest path tree rooted at s.

Indeed, one could use more flexible overlay reconfiguration operations to improve upon switch-trees and HostCast performance. Banerjee et al. [5] provide such a scheme. Simulation results (Section 3.2) show that our approach achieves better performance than their solution.

The next subsection describes the high-level working of the MeshTree protocol. We refer readers to [21] for a detailed treatment of the operations. Section 3.2 presents our simulation results.

3.1. Distributed MeshTree Overlay Construction

The main objective of the MeshTree protocol is to obtain the desired overlay structure: a degree-bounded mesh that contains a low cost backbone tree with shortcut links. The data delivery tree can then be obtained from the mesh as a reverse tree rooted at s using a path-vector routing protocol [8].

To achieve the overlay structure in a scalable manner, MeshTree uses the incremental improvement strategy typically used by distributed tree building protocols. First, the overlay grows when newcomers join in the session. Newcomers are randomly attached to the overlay. Thus, the initial overlay is unoptimised. Then, every MeshTree node (except the source, s) periodically tries to improve its own local overlay structure. Each improvement process involves adding/deleting links to/from the overlay using only the topology knowledge at the nodes involved.

The following discussion sketches the working of the improvement process. In the overlay, every node maintains a set of overlay neighbours (within their capacity constraint). A unicast connection (i.e. overlay link) is formed between each pair of neighbours. Basically, the overlay links can be classified into three groups:

- Backbone tree links. These are links included in the backbone tree.
- Delivery tree links. These are links included in the delivery tree. Note that a backbone link can also be a delivery tree link, and vice versa.
- Pure mesh links. These are links that are neither in the backbone nor the delivery tree.

In the improvement process, a node periodically tries to locate and add a new overlay link that provides substantial performance gain. If adding the link will result in a degree violation to the nodes involved (i.e. the two end points of the link), an existing link may be dropped. To achieve the above, a node say x needs to: (i) identify a potential neighbour; and (ii) establishing the overlay link to the potential neighbour.

Identify a potential neighbour First, x forms a candidates list. The candidates are selected from nodes within a small overlay distance of x, and other non-neighbour members acquired with a gossip-style node discovery (see [21]). Node x then performs distance measurements to the candidates. From the measurements, x will pick the node that gives the most reduction in the backbone tree cost. If no such node exists, x will pick the node that most improves the delivery tree delay. Otherwise, a node is randomly picked from the list. Say y is picked as the potential neighbour.

Establish the overlay link Node x needs to initiate a peering request to y. An overlay link will be created if x and y can reach a common consensus about their neighbouring relationship, i.e. whether the new link is a backbone tree and/or delivery tree or a pure mesh link.

The main consideration at y is its capacity constraint. It will accept x if it still has spare capacity. Otherwise, it will determine if an existing neighbour can be dropped in favour of x, using a set of local rules. The rules make sure that the dropped neighbour will not: (i) partition the delivery tree; (ii) increase the backbone tree cost; and (iii) increase the delivery tree delay.

3.2. Performance Evaluation

We conduct simulation experiments on a set of Transit-Stub topologies generated using the GT-ITM topology generator [2] as well as the power-law topologies generated by the Inet generator [1]. We report representative results obtained from a 10100 nodes transit-stub network. (Similar performance trends were observed in the power-law topologies.)

We first compare the quality of the delivery trees built by MeshTree against the following two schemes.

- *Compact Tree Algorithm (CPT)* [19]: CPT is a centralised greedy heuristic to create minimum delay degree-bounded trees.
- *Banerjee et al.'s scheme* [5]: This is a distributed iterative overlay tree improvement scheme. It has been shown to outperform several other distributed protocols: switch-trees, HostCast, HMTP, AOM and TBCP in the delay property [20]. The scheme defines a set of local transformations such as parent switching and swapping to reconfigure nodes

within two levels of each other. Nodes also perform random swapping with low probability so as to avoid local minima.

The quality of the overlay tree is judged by the following metrics: (i) *RMP and RAP*; (ii) *tree cost ratio (TCR)*; and (iii) *link stress*. RMP and RAP are two variants of relative delay penalty [8]. RMP (RAP) is the ratio between the maximum (average) overlay delay and the maximum (average) delay using unicast from s to all other nodes. Hence, RMP represents our optimisation objective, while RAP indicates the average delay observed by the receivers; Tree cost [25] is defined as the sum of delays on the tree's links. It provides a simplified view of the total network resource consumption of a tree. TCR is the ratio of the cost between an overlay tree and the corresponding network layer multicast tree; Finally, link stress is defined as the number of copies of an identical packet sent over a single link.

In the experiments, all members randomly join in the session within the first 50s. The first member automatically becomes the data source. The out-degrees of the overlay nodes are uniformly distributed between 2 and 10. Both MeshTree and Banerjee et al.'s scheme use an improvement period of 30s, and the results are collected after the trees converge.

In terms of delay performance (RMP and RAP as in Fig. 2(a) and (b)), we can see that MeshTree always outperforms Banerjee et al.'s scheme. For group sizes from 32 to 256, it produces trees with lower RMP and similar RAP compared with CPT. For larger group sizes where we expect a centralised approach to be unsuitable, MeshTree still shows reasonably good delay properties.

Fig. 2(c), (d) and (e) depict the worst-case and average link stress, and the TCR performance. We can now observe that CPT produces low delay trees at the expense of high traffic redundancy and network resource usage. The fact that its worst-case stress grows rapidly also suggests that it is not suitable for larger group sizes. MeshTree shows a much lower maximum stress performance, which is close to that of the Banerjee et al.'s scheme. In addition, it always shows the lowest average link stress and tree cost properties.

We also conducted experiments where the source uses a direct unicast connection to each of the receivers. Obviously, this has the best delay performance. However, it overloads the source node, and results in a worst-case stress that is as high as the group size. It also incurs much higher resource usage, for example, its TCR is about 16.5 for a group size of 1024 members — an order higher than the ALM solutions.

In Fig. 2(f), we examine the convergence property of MeshTree. The figure shows the evolution of the TCR of the backbone tree, the RAP and RMP of the delivery tree, for a 1024-node overlay. We can see that these values increase quickly as nodes are joining the overlay. This is because the initial overlay is randomly connected. In the experiment, each receiver has a improvement period of 30s. We can see that the RAP and RMP values rapidly decrease to a value less than 2 within the first 200s, i.e. less than 10 improvement rounds per node. This indicates that MeshTree can converge very quickly. The result also shows that MeshTree can achieve low backbone tree cost ratio, which suggests that the overlay contains a lot of short links between the members. This helps to reduce the delivery tree cost and link stress, as observed previously.

The high delay and cost at the early stage is obviously undesirable. While not shown here, we have found that this can be greatly improved by using a larger number of joining targets. For example, we can achieve a 50% improvement by using two initial join targets per node.

We also investigate several other properties of MeshTree (The results are omitted due to space consideration). In our experiments with overlays up to 1024 nodes, we found that the protocol overhead observed per overlay node is reasonably low, i.e. less than 1kbps. We also study the tree restoration speed when nodes depart from the overlay. The results show that MeshTree can quickly reconstruct the delivery tree following failure faster than the schemes studied in [24].

4. Concluding Remarks

This paper presents MeshTree, a distributed delay-optimised degree-bounded overlay multicast tree construction protocol. Using simulation experiments, we show that the trees built with MeshTree have delay properties comparable with (and sometimes better than) a centralised algorithm, and always have lower delay than a decentralised scheme. Both alternative schemes compared have previously been shown to perform well in their class. In addition, trees constructed with MeshTree consume fewer network resources. MeshTree also exhibits some properties that are important as a distributed solution, i.e. low protocol overhead, fast convergence and fast failure recovery speed.

References

[1] Inet topology generator, available at: topology.eecs.umich.edu/inet.

[2] GT-ITM topology generator, available at: www.c.gatech.edu/Ellen.Zegura/graphs.html.

[3] The hypercast project, http://www.cs.virginia.edu/ mngroup/hypercast/.

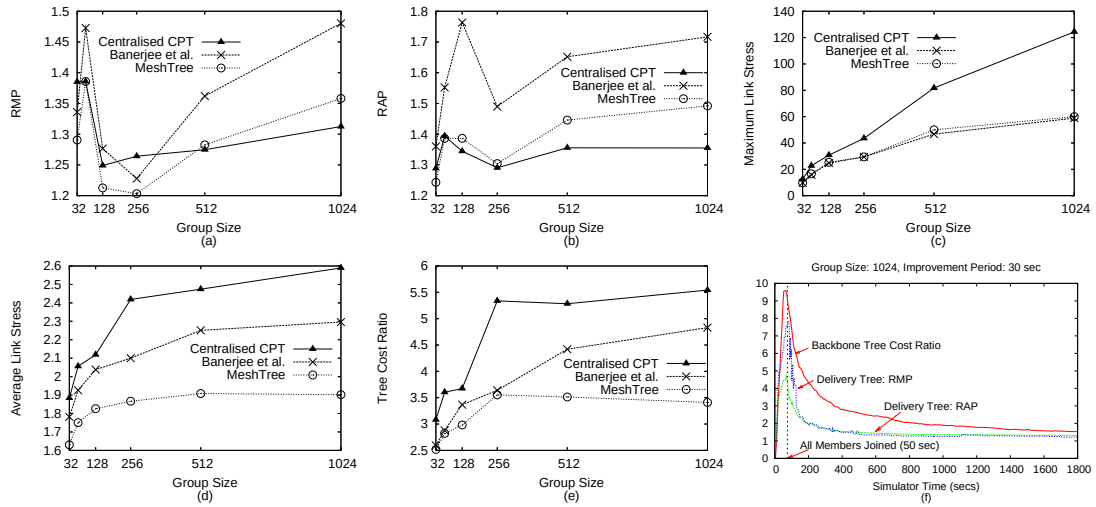


Figure 2. Evaluation Results: (a) RMP; (b) RAP; (c) Maximum link stress; (d) Average link stress; (e) Tree cost ratio; (f) Convergence property

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM*, pages 205–220, Pittsburgh, PA, 2002. ACM.

[5] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *IEEE INFOCOM*, San Francisco, USA, 2003.

[6] M. Casto, P. Druschel, A. M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct 2002.

[7] Y. Chawathe. *An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, 2000.

[8] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, Santa Clara, CA, 2000.

[9] A. El-Sayed, V. Roca, and L. Mathy. A survey of proposals for an alternative group communication service. *IEEE Network*, 17(1):46–51, 2003.

[10] P. Francis, Y. Pryadkin, P. Radoslavov, R. Govindan, and B. Lndell. Yoid: Your own internet distribution. Unpublished, ISI, 2000.

[11] D. A. Helder and S. Jamin. End-host multicast communication using switch-trees protocols. In *GP2PC*, 2002.

[12] J. Konemann. *Approximation Algorithms for Minimum-cost Low-degree Subgraphs*. PhD thesis, CMU, 2003.

[13] Z. Li and P. Mohapatra. Hostcast: A new overlay multicast protocol. In *IEEE ICC*, Alaska, USA, 2003.

[14] L. Mathy, R. Canonico, and D. Hutchison. An overlay tree building control protocol. In *NGC*, London, UK, 2001.

[15] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, San Diego, Carlifornia, 2001.

[16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC*, London, UK, 2001.

[17] S. Ratnasamy, S. Shenker, and I. Stoica. Routing algorithms for DHTs: Some open questions. In *1st IPTPS'02*, Cambridge, MA, 2002.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware*, 2001.

[19] S. Y. Shi, J. S. Turner, and M. Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *NOSSDAV*, New York, USA, 2001. ACM.

[20] S. W. Tan, G. Waters, and J. Crawford. A study of distributed low latency application layer multicast tree construction. In *LCS'04*, London, UK, 2004.

[21] S. W. Tan, G. Waters, and J. Crawford. MeshTree: A delay-optimised overlay multicast tree building protocol. Technical Report 5-05, Computing Laboratory, University of Kent, April 2005.

[22] D. A. Tran, K. A. Hua, and T. T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *IEEE INFOCOM*, San Francisco, USA, 2003.

[23] S. Wu, S. Banerjee, X. Hou, and R. A. Thompson. Active delay and loss adaptation in overlay multicast tree. In *IEEE ICC*, Paris, France, 2004.

[24] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *IEEE INFOCOM*, H.K., 2004.

[25] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *IEEE INFOCOM*, pages 1366–1375, New York, USA, 2002.