DISTRIBUTED CONSTRUCTION OF RESOURCE-EFFICIENT OVERLAY TREE BY APPROXIMATING MST

Yuan Li
School of Computing
National University of Singapore,
email: liyuan@comp.nus.edu.sg

Wei Tsang Ooi
School of Computing
National University of Singapore,
email: ooiwt@comp.nus.edu.sg

ABSTRACT

This paper presents a distributed protocol called RESMO for constructing overlay tree to support video streaming applications. RESMO reduces network resource usage by approximating minimum spanning tree and achieves low end-to-end latency between the sender and each receiver at the same time. The resulting overlay is a compromise between overlay minimum spanning tree and shortest path tree. We evaluated the tree constructed by RESMO through simulations, and found that RESMO gives significant improvement over existing protocols in terms of link stress, relative delay penalty and resource usage.

1. INTRODUCTION

IP multicast allows efficient one-to-many data delivery by eliminating data duplicates on network links. However, due to scalability issues, it is still not widely deployed. In recent years, many application-layer multicast (ALM) protocols [1–6] have been proposed as an alternative method for one-to-many communication. In ALM, routing and data forwarding is carried out at the application layer instead of the network layer. The multicast tree in ALM (also known as *overlay tree*) is a virtual delivery tree built on top of underlying network where each edge consists of a unicast route between two overlay nodes. Unlike IP multicast, ALM may introduce duplicate packets on physical links. Chu et. al. [2] defined the following metrics to evaluate the efficiency of overlay trees: (a) *link stress*: number of duplicate packets carried by each link, (b) *resource usage*: $\sum_{i=1}^{L} d_i * s_i$ where L is the number of active physical links covered by the overlay tree, d_i is the delay of link i and s_i is the link stress of link i, and (c) *relative delay penalty* (RDP): the ratio of the delay between the source and a receiver along the overlay tree to the unicast delay between them.

In order to disseminate real-time multimedia data efficiently, an overlay tree needs to have low end-to-end delay between the sender and each receiver. Furthermore, due to the bandwidth intensive nature of multimedia data, constructing a tree with low resource usage is important. Since resource usage is equivalent to the sum of the virtual edge delays in an overlay tree, a minimum spanning tree (MST) has minimum resource usage. However, an MST does not guarantee minimum end-to-end delays. Several existing work have studied constructions of overlay trees that trade off between resource usage and end-to-end delays [6, 7].

This paper presents a distributed protocol called RESMO (**R**esource **E**fficient **S**calable **M**ulticast **O**verlay) that builds an overlay tree connecting overlay nodes by *approximating* an MST. RESMO sacrifices the minimality of resource usage in order to improve end-to-end delays. We show that RESMO has resource usage comparable to MST, while keeping RDP low. Our simulations indicates that the RDP and link stress of RESMO are lower than those of NICE [4] and Narada [2], two reputable ALM protocols. RESMO also shows good scalability behavior when group size increases.

The rest of the paper is organized as follows. Section 2 discusses the related work. RESMO protocol is presented in Section 3. We explain and analyze the simulation results in Section 4. Section 5 concludes the paper.

2. RELATED WORK

The existing ALM protocols can be generally classified into two categories: tree-first (ALMI [1], HMTP [5], NICE [4]) and mesh-first protocols (Narada [2], Gossamer [3]). Narada builds the overlay tree on top of a mesh. The tree's efficiency depends on the mesh's quality. RESMO is similar to Narada as it is a mesh-first protocol. However, Narada uses DVMRP to build a shortest path tree, while RESMO tries to reduce resource usage by approximating MST. NICE [4] is a hierarchical clustering-based protocol which is designed for low bandwidth application with large group size. In contrast, RESMO is non-hierarchical, and is designed with small to medium size group in mind for low resource usage and low end-to-end delay.

RESMO is similar to PBDT [6] in the way that both protocols aim to trade-off MST with SPT. But in PBDT, the sender assigns a priority to each receiver with respect to their application-layer features and then uses this priority to

0-7803-8603-5/04/$20.00

balance between delay and resource usage. It is a centralized protocol. In contrast, RESMO is completely distributed and assumes a single priority for each receiver. LAST [8] is another algorithm that trades off latency with cost. This approach traverses a MST in a depth-first fashion, whenever it encounters a node with MST delay larger than SPT delay by a treshold, it adds links from the node's shortest path to the current tree. LAST requires construction of a MST first before constructing its distribution tree. On the other hand, RESMO constructs an approximation of MST as its distribution tree directly.

3. PROTOCOL DESCRIPTION

RESMO is a mesh-first protocol – nodes in RESMO maintains a mesh and the overlay tree is build on top of the mesh. Since our focus in this paper is the construction of the overlay tree, we do not elaborate on the orthogonal issues of mesh construction and maintenance mechanism. In our simulation, we simply use expanded ring search to discover neighboring nodes to form the mesh. Only links with sufficient bandwidth to support the current session requirement are considered in the mesh.

RESMO constructs the overlay step-by-step, "growing" the tree from the sender. At each step, the current leaf nodes in the tree are actively involved in constructing the tree by sending out invitations to its neighbors to join the tree. This process stops when a leaf node has no neighbors that is interested in the given session.

A RESMO node can be in one of the following five states (with respect to a session) at one time: (a) *sleep*: a node is in this state if it has not been invited to join the session. (b) *awake*: a node is *awake* if it has received one invitation to join the tree, but has not yet become part of the tree. (c) *weighing*: a node is in the *weighing* state if it has received multiple invitations to join the tree, and is deciding which inviter should be its parent. (d) *inviter*: a node is an *inviter* if it has a parent but has no children. Overlay nodes in this state will send invitations to its neighbors to expand the overlay tree, constructing the tree recursively. (e) *parent*: a node is said to be in the *parent* state when it has one or more children.

Each node maintains a soft-state table called *rtttable* that records the round-trip time to each of its neighbors in the mesh. The table is maintained by exchanging alive messages occasionally, and is useful in selecting parents and tree partition recovery.

We now describe how the tree is built in details.

The tree construction process begins when a sender S sends an **invite** message to its neighbors, which are in *sleep* state. Each neighbor of S, upon receiving **invite**, replies with a **thanks** message and goes into the *awake* state.

S will add the first neighbor that replies as its first child, and ignore other **thanks** messages that arrive later. Let this first neighbor be G_0. S sends a **be-my-child** message to G_0 and goes into the *parent* state.

After receiving **be-my-child** message from S, G_0 sets S as its parent, and switches from *awake* to *inviter* state. G_0 now sends **invite** messages to its neighbors, but only to those that have not been involved in the tree construction.

It is possible that a node G receives multiple **invite** messages while it is *awake*. This happens when G is a common neighbor of some inviter nodes. In this case, G should decide which one of these neighbors should be chosen as its parent. G sets a timer $T_{wgh,j}$ for each inviter G_j. G is said to be in the *weighing* state. When timer $T_{wgh,j}$ expires, G examines its *rtttable* and picks the inviter with smallest RTT as its parent by sending a **be-my-parent** message to the chosen parent. Then G cancels all the other pending timers and goes from *weighing* state to *inviter* state.

The selection of parent with smallest RTT is crucial to the approximation of MST in RESMO. The value of $T_{wgh,*}$ can be tuned to trade-off resource usage with RDP of the resulting overlay tree. A large $T_{wgh,*}$ allows G to wait for more invitations, hence increasing the chance of selecting a parent with lower RTT. While selecting parent with lowest RTT reduces resource usage, it does not guarantee that the end-to-end delay from S to G is also small. The key to reducing end-to-end delay is to use smaller $T_{wgh,*}$. Since RESMO constructs the tree in a stepwise fashion, a node that has lower delay from the source (measured along the constructed tree) is more likely to become an inviter and send out invitations sooner. By using smaller $T_{wgh,*}$, a node will select those who become inviters sooner and have lower delays from the source, thus reducing the end-to-end delay from S to itself.

A node G always set a time T_{ivt} when it receives an invitation for the first time, say, from an inviter G_l. If G is still in the *awake* state when the timer expires (i.e., no other invitations has been received by G), then it is probable that other routes from S to G has long delays. In this case, G asks to join the tree by sending **be-my-parent** to G_l. By adjusting the value of T_{ivt}, we can tune the tolerable end-to-end delays in the tree.

Due to space constraints, we do not elaborate on tree update mechanisms here. Briefly, when a node joins the session or needs to look for a new parent (either because the parent has left or failed), it finds the neighbor with lowest RTT from its *rtttable* and sends **be-my-parent** message to that neighbor.

As network conditions and group memberships change continuously, RESMO must adapt since the overlay tree may not be efficient anymore. We currently reconstruct the tree periodically. We are developing mechanisms that allow incremental updates of the tree to avoid the expensive overhead in reconstructing the tree.

4. SIMULATION AND EVALUATION

Using simulations, we (i) determined the effects of weighting timer on the constructed tree, (ii) compared the performance of RESMO with five other schemes, namely NICE, Narada, MST, SPT and Unicast, using RDP, resource usage, link stress and maximum node degree as comparison metrics. Simulations of NICE and Narada are based on the *myns* simulator provided by the authors of NICE, while the rest of the schemes are simulated in ns-2.

Using GT-ITM toolkit, we randomly generated ten 1000-nodes transit-stub topologies with 0.42 edge connection probability within stub domains. Overlay nodes are randomly located in the stub domains and the sender is randomly selected from the overlay nodes. In each experiment, session group size is varied between 25 and 150 to evaluate the scalability of RESMO. The results are averaged over these ten topologies.

In our experiments, we set the value of T_{ivt} to $8d$, where d is the estimated one-way delay (obtained from *rtttable* between the inviter and the receiving node). $T_{wgh,*}$ is set to $k \times d$, for some constant k. To find reasonable values of k, we vary $T_{wgh,*}$ between $0.2d$ and $6d$ and plot the different performance metrics for group size 100 (we found similar curves for other group sizes). As expected, increasing k reduces average link stress (Figure 1), resource usage (Figure 2), but it also deteriorates RDP (Figure 3) and increases the tree construction time. Values of k between 2 and 4 shows a good compromise between the various performance metrics. We therefore set $T_{wgh,*}$ to a uniform random value between $[2d, 4d]$.

Our other findings can be summarized as follows:

(a) Link stress of RESMO is significant lower than those of NICE and Narada and is comparable to MST. Figure 4 shows that for each group size, RESMO has around 1.46 mean stress, which is about 50% of NICE (2.33 - 3.30) on average although they have similar number of active physical links. This is because RESMO is a fully distributed protocol where each node shares the forwarding task evenly; whereas NICE is a hierarchical protocol and leaders at each level will forward more packets. The large confidence interval of NICE supports our analysis. The mean link stress of RESMO is about 60% of Narada (2.01 - 3.02) on average. Narada uses DVMRP to produce lower delay, thus it uses fewer number of physical links which in turn introduces more link stress.

(b) RESMO has RDP values between those of MST and SPT. For 100 nodes topology, the 90% RDP for these trees are: 2.0 (RESMO), 2.8 (MST), and 1.4 (SPT) respectively (see Figure 5). We also compare the RDP with NICE and Narada: NICE has more overlay nodes with RDP less than 1.4 than other schemes, but the 90% RDP is up to 4.0 which is twice of RESMO. Furthermore, increasing the group size

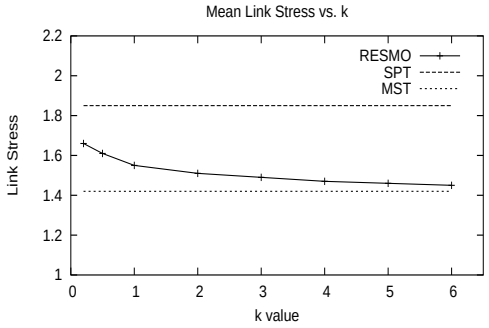

Fig. 1: Mean Link Stress with 90% Confidence Interval vs. k.

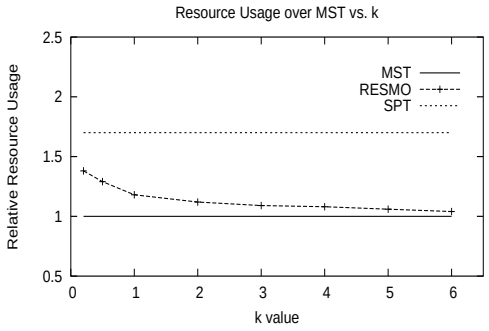

Fig. 2: Resource Usage over MST vs. k.

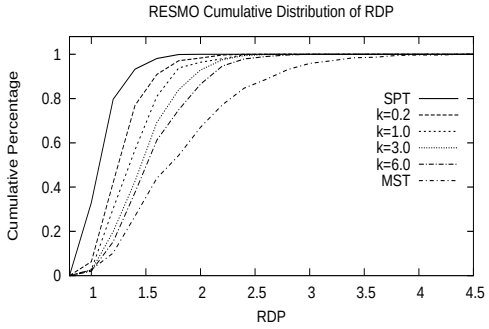

Fig. 3: Cumulative RDP vs. k.

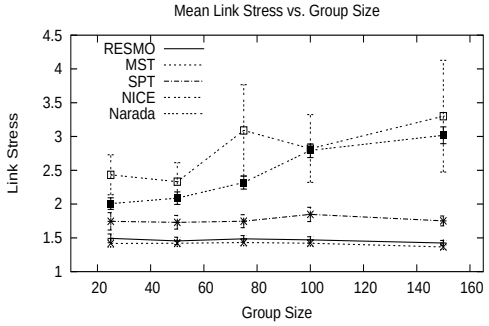

Fig. 4: Mean Link Stress with 90% Confidence Interval.

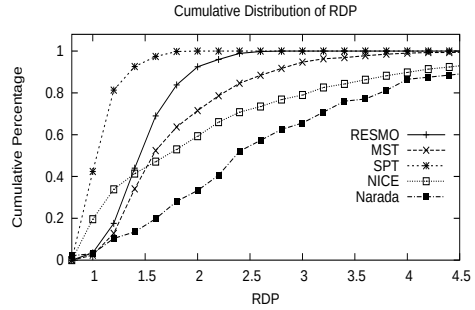

Fig. 5: Cumulative RDP of Group Size 100.

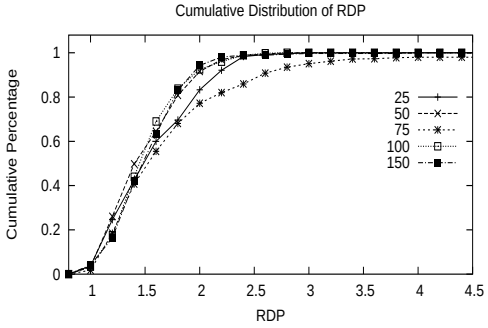

Fig. 6: Cumulative RDP with Different Group Size.

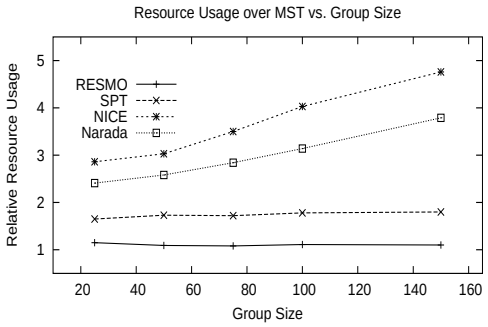

Fig. 7: Resource Usage over MST.

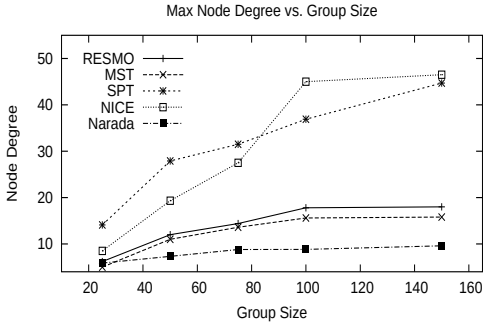

Fig. 8: Maximum Node Degree.

from 25 to 150 does not degrade RDP of RESMO (Figure 6).

(c) RESMO also has resource usage comparable to MST (9%-15% more, see Figure 7). NICE and Narada have much higher resource usage, possibly due to the existence of multiple paths in both protocols. When the group size increases, their relative resource usage increases also while the relative resource usage of RESMO remains stable.

(d) The maximum node degree of RESMO is close to MST and slightly increases with the group size (Figure 8). Narada has lowerest maximum node degree due to the degree constrains in its protocol.

5. CONCLUSION AND FUTURE WORK

We have presented a distributed protocol for building resource-efficient overlay tree by approximating MST. The resulting tree is comparable to MST in resource usage, link stress and node degree, but has low end-to-end latency as well. We are currently improving RESMO to reconstruct the tree incrementally under changing network conditions, and we plan to implement RESMO over the wide-area experimental platform PlanetLab to study its performance in real network settings.

6. REFERENCES

[1] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An application level multicast infrastructure," in *Proc. of 3rd Usenix Symp. on Internet Technologies and Systems*, Mar. 2001.

[2] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," in *Proc. of ACM SIGMETRICS*, June 2000.

[3] Y. Chawathe, "Scattercast: An architecture for internet broadcast distribtuion as an infrastructure service," Ph.D. dissertation, University of California, Berkeley, Dec. 2000.

[4] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proc.of ACM SIGCOMM*, Aug. 2002.

[5] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," in *Proc. of IEEE infocom*, June 2002.

[6] J. Vogel, J. Widmer, D. Farin, M. Mauve, and W. Effelsberg, "Priority-based distribution trees for application-level multicast," in *Proc. of NetGames 2003*, May 2003.

[7] D. Kostic and A. Vahdat, "Latency versus cost optimizations in hierarchical overlay networks," *Duke Technical Report*, Nov. 2001.

[8] S. Khuller, B. Raghavachari, and N. Young, "Balancing minumum spanning and shortest path trees," *Algorithmica*, vol. 14(4), pp. 305–321, 1994.