Online Network Coding for the Dynamic Multicast Problem

Fang Zhao and Muriel Médard
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology, Cambridge, MA 02139, USA
E-mail: {zhaof, medard}@mit.edu

***Abstract*— Many of the multimedia applications such as video broadcasting and teleconferencing require the network to support dynamic multicast sessions when the membership of the multicast group changes over time. In this paper, we study this problem in the context of coded networks. While trying to minimize the cost of the multicast, we also want to minimize the disturbances to existing users when the multicast graph changes. To characterize disturbances to users, we define two types of rearrangements, link rearrangements and code rearrangements. We present four algorithms to solve the non-rearrangeable and rearrangeable versions of the dynamic multicast problem. Simulation results show that the α-scaled algorithm we proposed can keep the cost of the multicast close to that of the optimal solution when the multicast subgraph evolves, and, at the same time, causes very few rearrangements during the process.**

I. Introduction

Network coding has been shown to be able to improve the throughput of multicast connections in a network [1], [2]. Ho *et. al.* [3] have shown that once a subgraph that satisfies the Min-Cut Max-Flow bound for each source destination pair is found, random coding can be used to multicast on this subgraph. In addition, the problem of establishing the optimal multicast subgraph has been studied by several researchers [4], [5]. While finding min-cost multicast tree in traditional routing networks requires solving the Steiner tree problem, which is NP-complete, min-cost multicast subgraphs in coded networks can be found by solving a linear programming (LP) problem, and this can be done in a distributed manner [6].

The works mentioned above focus on the problem of static multicast, where a connection is setup for the use of a multicast group, whose membership stays constant throughout the connection duration. On the other hand, in applications such as real-time video distribution and teleconferencing, users can join or leave the multicast group at any time during the session. In such cases, we need to adjust the multicast subgraph to cater for the needs of this dynamic group. Lun *et. al.* gave a dynamic programming formulation of this problem in [7], which aims to deliver continuous service to the users. However, link and code rearrangements, which are defined later, can still occur under their formulation.

In the context of traditional routing networks, this problem corresponds to the the dynamic Steiner tree (DST) problem [8]. In DST, it is important to limit the number of rearrangements as a connection evolves, because rearranging a large multipoint connection may be time consuming and may require significant use of network resources in the form of CPU time. In addition, rearrangement of a connection may result in the blocking of some parts of the connection as rearrangement proceeds. Therefore, the DST problem comes in two flavors [8], [9]. One is the nonrearrangeable version, in which rearrangement of existing routes is not allowed. In the other version, rearrangement is allowed, but the cost of rearrangements is taken into consideration.

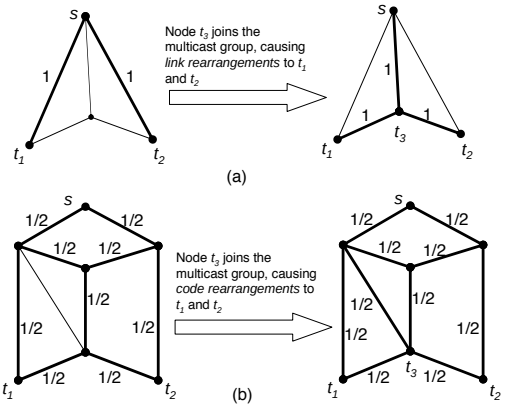

Fig. 1. (a) Example of an online step that causes link rearrangements to existing users; (b) Example of an online step that causes code rearrangements to existing users. The multicast rate from source node s to terminal nodes $\{t_1, t_2, t_3\}$ is 1. The thick lines indicate links used in the multicast, and the numbers against them indicate the rate of flow on them.

The situation is similar in networks with coding. When the membership of the multicast group changes, we want to minimize the disturbance to existing users by limiting both *link rearrangements* and *code rearrangements*. A *link rearrangement* occurs when some links in the current multicast subgraph is removed causing alternate paths to be used to serve existing users (see Fig. 1(a) for an example). Like in the routing networks, owing to the change in the physical connection, this kind of rearrangement causes disruptions to the continuous service to the multicast group. The second kind of rearrangement, which we call *code rearrangement*, is more subtle. Code rearrangement occurs when new incoming links are added to existing nodes in the multicast subgraph (see Fig. 1(b) for an example). Since random coding is used by the intermediate nodes in the subgraph to perform network coding, when a node has an additional incoming link, it will

have to generate a new set of random parameters to mix the incoming streams. All receivers downstream, therefore, have to use these new parameters and recompute the inversion matrix to decode the data streams. This scenario does not involve any physical switching of paths for the existing terminals, but it still causes a minor disruption to the continuous service due to this reprocessing of network coding parameters. Note that the disruptions caused by code rearrangements are generally smaller than that caused by link rearrangements.

For the dynamic multicast problem, there are two extreme cases. On the one hand, we can simply use the new optimal subgraph whenever there is an update to the multicast group. In this case, users in the group will experience a lot of disruptions, but the cost of the multicast is always kept minimal. On the other hand, we can enforce that no link or code rearrangement is allowed for all existing users throughout the multicast session. In this case, users enjoy uninterrupted services, but in general, the subgraph used will deviate further and further away from the optimal one. In this paper, we present one algorithm to solve the nonrearrangeable version of the dynamic multicast problem, and three algorithms for the rearrangeable version. Simulation results show that one of the rearrangeable algorithms, the α-scaled algorithm, can be used to strike a balance between cost and frequency of user disturbances in a distributed manner. Although we present our algorithms based on wireline networks, they can be easily extended to wireless networks as in [4].

The rest of this paper is organized as follows. In Section II, we present the LP formulation of the static multicast problem, and extend it to solve the nonrearrangeable version of the dynamic problem. In Section III, we introduce three algorithms that allow some rearrangements but are simpler and more practical. Simulation results for the algorithms proposed are presented in Section IV, and finally, the paper is concluded in Section V.

II. Nonrearrangeable Algorithm

In this paper, we focus on the problem of dynamic multicast in wireline coded networks, where a source node s transmits to a group of terminal nodes T, and this group changes over time. We refer to each change of the membership of T (either an addition or a removal of a terminal node) as an *online step*. We model the network by the directed graph $G = (N, A)$, where N is the set of nodes and A is the set of communication links. Let the cost of transmitting on link (i, j) be $a_{ij}z_{ij}$, where z_{ij} is the rate of flow on this link. For the static multicast problem, Lun *et. al.* [6] has given an LP formulation as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in A} a_{ij} z_{ij} \\
\text{subject to} \quad & z_{ij} \geq x_{ij}^{(t)}, \ \forall (i,j) \in A, t \in T, \\
& \sum_{\{j|(i,j)\in A\}} x_{ij}^{(t)} - \sum_{\{j|(j,i)\in A\}} x_{ji}^{(t)} = \sigma_i^{(t)} \qquad (1) \\
& \forall i \in N, t \in T, \\
& 0 \leq x_{ij}^{(t)} \leq c_{ij}, \ \forall (i,j) \in A, t \in T,
\end{aligned}$$

where $\sigma_i^{(t)}$ is R if $i = s$, $-R$ if $i = t$, and 0 otherwise, and c_{ij} is the capacity of link (i, j). This LP problem can be solved by a number of methods. For example, Simplex method can be used to solve it centrally, and subgradient method can be used to solve it distributedly when there are no link capacity constraints. When capacity constraints are present, we can approximate the cost function by a strictly convex function that grows sharply when approaching the link capacity, then use the distributed algorithm proposed in [6] to solve it. For simplicity, in this paper, we assume that the rate of the multicast is lower than the capacity of the links, which is generally the case in wireline networks. From here on, we denote any centralized/distributed algorithm that solves the LP problem as LP_{cent}/LP_{dist} respectively.

For the dynamic multicast problem, the initial multicast subgraph is setup by solving the above LP problem. To solve the nonrearrangeable version of this problem, we have to prevent link and code rearrangements from happening. To meet the no link rearrangement requirement, we basically need to make sure that the existing users still use the same path(s) for the multicast when the set T changes. This can be achieved by setting the cost of the links in the current subgraph, G_c, to zero. If the capacity of a link is larger than the rate used for the multicast, then the link is split into two virtual links, one with capacity equal to the rate used for the multicast and cost zero, and the other with the remaining capacity and cost unchanged. For example, if link (i, j) has capacity $c_{ij} = 2$ and rate of flow $z_{ij} = 1$ for the multicast, then nodes i and j treat link (i, j) as two parallel links with capacities 1 each, and one of them has cost of 0, and the other one has cost of a_{ij}. After doing this, the current multicast subgraph becomes "free", and doing optimization on this new cost assignment will always lead to using the same path(s) to serve the existing users in the new subgraph. Therefore, link rearrangements are avoided.

One problem with the above method is that some links not necessary for the new terminal set might be included in the new subgraph after a removal of a terminal. This is because all link in the old subgraph are free, and some of these links might still be included in the solution to the LP problem even though they are not necessary in performing the multicast to the new terminal set. To solve this problem, instead of setting their costs to 0, we can set the cost of the used links to a small value ϵ, so that no extra link would be included in the optimal solution, and, at the same time, the used links are still almost free as compared to the other links.

As for code rearrangements, we want to prevent the usage of new links that go into existing nodes of the subgraph. To do that, each node in the subgraph scans through its incoming links, and sets the cost of those unused ones to a very large value, M. Again, if the capacity of a link is not fully used in the current subgraph, we can split it into two parallel virtual links as above. These nodes then send the new costs of its incoming links to their corresponding tail nodes, and the new high costs can prevent these links from being used.

After making these changes to the link costs, when an online step occurs, we can simply run LP_{dist} again with the new

```
node i

nodeUsed = 0
for all (j,i) ∈ A
   if (j,i) ∈ G_c
      a_ji = ε
      nodeUsed = 1
   end
end
if nodeUsed = 1
   for all (j,i) ∈ A
      if (j,i) ∈ G_c     a_ji = M
   end
end
call LP_dist
```

Fig. 2. Nonrearrangeable algorithm.

```
node i

for all (j,i) ∈ A
   if (j,i) ∈ G_c      a_ji = ε
end
call LP_dist
```

Fig. 3. MLR algorithm.

costs, and obtain a feasible subgraph for the new multicast group without any link or code rearrangements. This algorithm is summarized in Fig. 2.

The above algorithm can be complicated due to the splitting of physical links into parallel virtual links. This requires more processing at the nodes and more coordination between the end nodes of the links. In addition, in the non-rearrangeable setting, it is inevitable that the subgraph used would deviate further and further from the optimal subgraph. This is because the no-rearrangement requirement forces the subgraph used to be as close as possible to the initial subgraph. Thus, when the multicast group moves away from the original group over time, our multicast subgraph becomes more and more suboptimal.

To simplify this algorithm and keep the cost of the multicast low, we may need to make some compromise and allow some rearrangements. In the next section, we present three such heuristic algorithms.

III. Rearrangeable algorithms

A. Algorithm for minimizing link rearrangement (MLR)

One way to simplify the nonrearrangeable algorithm is to focus on eliminating link rearrangement only, and ignore code rearrangement. This can be easily done by setting the used links costs to a very small value ϵ after each online step as in the nonrearrangeable algorithm, and call LP_{dist} to solve the new LP problem. This algorithm, which we call the MLR (minimal link rearrangement) algorithm, is shown in Fig. 3.

The motivation for this algorithm comes from the observation that the need for each node to keep track of unused incoming links arises when we have a non-tree subgraph, and things would be much simpler if we only have to deal with trees. This is because, in trees, each node only has one incoming link, and it has full information of the multicast. Thus, there is no worry about code rearrangement.

Notice that once the multicast subgraph becomes a tree, it will remain as a tree through the rest of the online steps.

```
call LP_cent to compute C_opt and G_opt
for all (j,i) ∈ A
   if (j,i) ∈ G_c      a_ji = ε
end
call LP_cent to compute C_nc and G_nc
if C_nc > C_opt × (1 + β)
   use G_opt for the multicast
else
   use G_nc for the multicast
end
```

Fig. 4. LMC algorithm.

To see this, consider addition of a new node to the multicast group. Since the original subgraph G_c is considered "free" and each node in G_c has full information of the multicast, the new terminal node only needs to find the shortest path from any node in G_c to itself, and attach itself to the subgraph. As for the removal of a terminal, a part of the tree may be removed, and the remaining graph should still be a tree. Since at every step, if G_c is not a tree, there is some positive probability that it will become a tree, and once it evolves into a tree, it will stay that way till the end of the multicast section. Therefore, if we keep running the dynamic multicast session, the probability that we are dealing with trees goes to 1.

In addition, simulations on practical networks show that in more than 98% of the time, we do get the optimum Steiner tree at startup. Therefore, we can focus on link rearrangements only and use the MRL algorithm. This algorithms still works if the initial subgraph is not a tree, the only difference is that we cannot guarantee that there will not be any rearrangements in such cases. Note that although network coding is not needed to perform multicast when the subgraph is a tree, our formulation of the problem allows us to find the optimal subgraph (be it a tree or not) in a distributed manner, whereas in traditional routing networks, find the Steiner tree is NP-complete.

B. Algorithm for limiting multicast cost (LMC)

If we use the MLR algorithm, it is expected that as time goes on, the subgraph used for multicast will move further and further away from the actual optimal subgraph for the current set of terminal nodes. As an alternative, we introduce occasional rearrangements in order to keep the cost of the multicast close to optimal. We use the LMC algorithm, shown in Fig. 4, to do this. In this algorithm, the nodes run two programs in parallel, one of which generates the subgraph with no rearrangement using the algorithm presented above. We call this subgraph the *no-change subgraph* G_{nc}, and the cost of this subgraph C_{nc}. The other program keeps track of the optimal subgraph, G_{opt}, for the current set of multicast terminals, and the cost of G_{opt} is C_{opt}. At each step, the cost of the two subgraphs are compared, and if C_{nc} is higher than C_{opt} by a certain factor, β, we switch to the optimal subgraph. Using this method, we can control the trade off between the frequency of disturbances to the users and the cost of the subgraph used for the multicast by changing the value of β. However, this method requires the nodes to keep track of two subgraphs, and centralized coordination is needed to compare the costs and make the nodes switch between two subgraphs simultaneously.

```
node i

for all (j,i) ∈ A
   if (j,i) ∈ G_c      a_ji = α × a_ji
end
call LP_dist
```

Fig. 5. α-scaled algorithm.

C. α-scaled algorithm

We now present a simple approximate algorithm that can trigger "auto-switching" between G_{nc} and G_{opt} in a distributed manner. Instead of assigning a very small cost to the used links as in the MLR algorithm, we can use a scaled value of the original cost, i.e., for an existing link (i, j) in the subgraph, we use αa_{ij} as its cost in the future computations as long as it stays in the subgraph, where α is a scaling factor between 0 and 1. If $\alpha = 0$, it is the same as the MLR algorithm; and if $\alpha = 1$, we will be using the optimal subgraph every time. We refer to this algorithm as the *α-scaled algorithm*, and it is shown in Fig. 5.

To see why this heuristic works and how the constants α and β are related, consider the case of removal of a terminal node. The LMC algorithm compares the values of C_{nc} and $(1+\beta)C_{opt}$, and picks the lower of the two. Since G_{opt} may overlap with the existing subgraph from before the online step, we assume the cost of this overlapping part of the subgraph is C_{ol} and the cost of the rest of the optimal subgraph is C_{others}. Thus, the comparison is equivalent to

$$\frac{1}{1+\beta} \times C_{nc} \gtrless C_{ol} + C_{others} \tag{2}$$

On the other hand, in the α-scaled algorithm, we are effectively choosing the lower cost between these two.

$$\alpha \times C_{nc} \gtrless \alpha \times C_{ol} + C_{others} \tag{3}$$

If we set α to $1/(1+\beta)$, we can see that equations (2) and (3) are very similar except the first term on the right hand side. By scaling the existing link costs by α, we can satisfy the requirement that the cost of the subgraph used never goes over $(1+\beta)C_{opt}$, but owing to the scaling factor α on C_{ol}, the approximate algorithm switches to the optimal subgraph more often than required by β. Using similar analysis, we have the same results for the case of addition of a terminal.

Thus, using an appropriate α to scale the costs of the used links, the optimization can trigger auto-switching between the two subgraphs, thus keeping the cost of the multicast low. In addition, we can make α a time-varying variable. In general, when a link is first added into the subgraph, it is likely that it will remain there for a while. However, the probability that the link remains in the optimal subgraph decreases with the online steps. To capture this characteristic, we can use a lower value for α for the first few online steps after a new link is added, and increase α gradually later on. Also, in a practical network, it may not be desirable to make back-to-back changes to the link connections, i.e., addition of a link to the multicast subgraph followed by an immediately removal of it in the next step. We can reduce the occurrence of such events by setting the α of new links to ϵ for a few steps before raising it to the normal value of $1/(1+\beta)$.

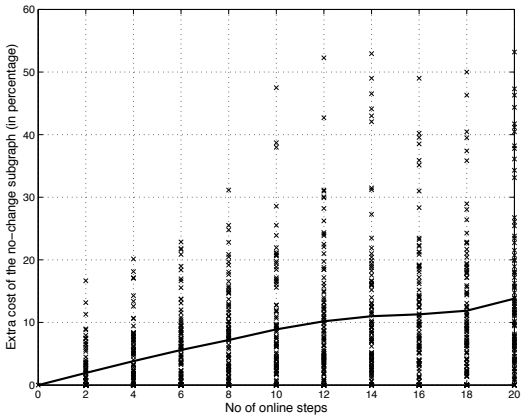

Fig. 6. Extra cost of the multicast subgraph generated by the MLR algorithm in terms of percentage of C_{opt} for the Exodus network. We are showing both the individual data points for each trial and the average curve.

IV. Simulation results

We first present simulation results for the MLR algorithm. The network topologies used in the simulations are obtained from the Rocketfuel project [10]. In each simulation, we start with a multicast from a random source to a set of 10 random terminals. Subsequently, in each online step, we first randomly decide whether there is an addition or removal of terminal, and then randomly select a terminal to add/remove based on that decision. Figs. 6 and 7 show the average increase of cost from the MLR algorithm as compared to C_{opt} in terms of percentage of C_{opt}. The network topology used for Figs. 6 and 7 are backbones for Exodus (US) and EBONE (Europe), respectively. As expected, the extra cost of the no-change subgraph grows approximately linearly with the online steps. In addition to the average curve, we also show the data points for each instance of the simulation in both Figs. 6 and 7. Note that there are cases when the cost of the no-change subgraph is as much as 60% higher than the optimal cost after 20 steps.

This undesirable phenomenon motivates the usage of the α-scaled algorithm. Fig. 8 shows the simulation results for using the α-scaled algorithm on the network used in Fig. 6. Here, we aim to control the cost of the multicast subgraph to be within $\beta = 30\%$ away from C_{opt}, thus, we use $\alpha = 0.75$. The average curve in Fig. 6 for the MLR algorithm is also shown here for comparison. The α-scaled algorithm provides lower cost for the multicasts as compared to the MLR algorithm. More importantly, the cost difference between the subgraph used and C_{opt} for the α-scaled algorithm is roughly constant after a while, and it does not grow over time. Of course, there is a price for this gain, which is the occasional switching from the no-change graph to the optimal graph. In this case, the average switching probability is 11.7%, which means, out of a hundred online steps, there are about 12 times when the existing users might experience distrubances to their transmissions.

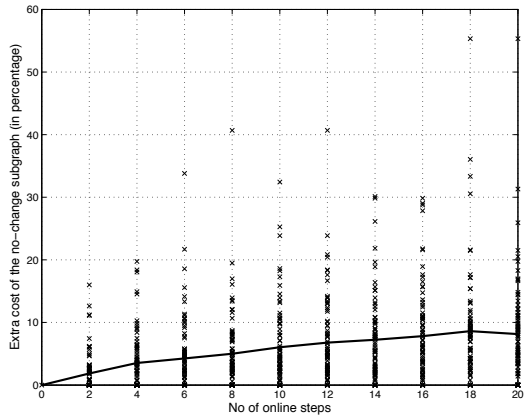

Fig. 7. Extra cost of the multicast subgraph generated by the MLR algorithm for the EBONE network in terms of percentage of C_{opt}.

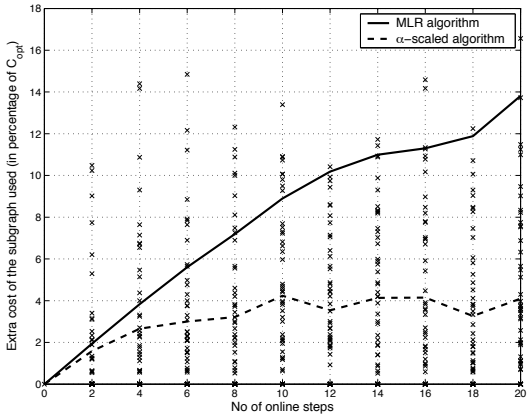

Fig. 8. Extra cost of the multicast subgraph generated by the α-scaled algorithm with $\alpha = 0.75$ and the MLR algorithm in terms of percentage of C_{opt}, on the Exodus network. We are also showing the individual data points for each trial for the α-scaled algorithm.

Furthermore, if we look closely at the data points for individual instances, we can see that, actually, none of the instance has gone over 20% higher than the optimal one. This is consistent with our discussion in Section III about the values of α and β. Therefore, if we want to have $\beta = 30\%$, we can use a lower value for α.

Finally, Fig. 9 shows the simulation results for the same network setup as Fig. 8 with different α values. As we can see, the higher the α value, the lower the average cost of the subgraph. At the same time, higher α values lead to higher switching rate. We observed that when α is equal to 0.5, the cost of the subgraph used is kept at around 9% higher than the optimal cost, whereas the switching probability is only 2.05%. Therefore, by selecting the α value properly, we can keep the cost of the multicast close to optimal during the multicast session while causing few disturbances to the existing users.

V. Conclusions

In this paper, we studied the problem of dynamic multicast in coded networks. In order to characterize the disturbances

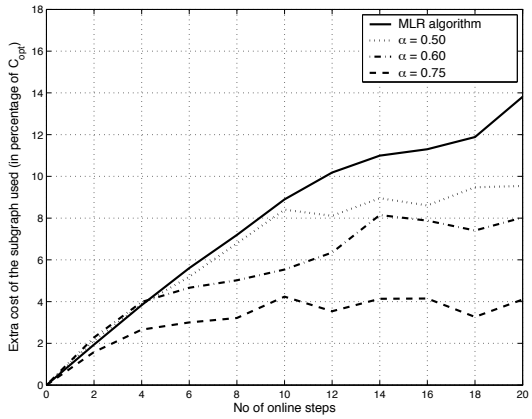

Fig. 9. Extra cost of the multicast subgraph generated by the α-scaled algorithm with various α values, on the Exodus network.

to users caused by the changes in the multicast subgraph, we introduced the concepts of link rearrangement and code rearrangement. We proposed both nonrearrangeable and rearrangeable algorithms for the dynamic multicast problem, and used simulation results to show that the α-scaled algorithm can effectively bound the growth of the multicast cost without causing too many disturbances to existing users.

Acknowledgment

This work was supported by the National Science Foundation under grant nos. CCR-0093349 and CCR-0325496, by Hewlett-Packard under grant no. 008542-008, and by the Air Force Office of Scientific Research under grant no. F49620-01-1-0365.

References

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Informaiton Theory*, vol. 46, no. 4, pp. 1204–1216, July 2000.

[2] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, October 2003.

[3] T. Ho, M. Médard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in *Proceedings of the 41th Annual Allterton Conference on Communication, Control, and Computing*, October 2003.

[4] D. S. Lun, M. Médard, T. Ho, and R. Koetter, "Network coding with a cost criterion," in *Proceedings of International Symposium on Information Theory and its Applications (ISITA)*, October 2004.

[5] Y. Wu, P. A. Chou, and S.-Y. Kung, "Minimum-energy multicast in mobile ad hoc networks using network coding," *IEEE Transactions on Communications*, vol. 53, no. 11, pp. 1906–1918, November 2005.

[6] D. S. Lun, N. Ratnakar, R. Koetter, M. Médard, E. Ahmed, and H. Lee, "Achieving minimum cost multicast: A decentralized approach based on network coding," in *Proceedings of IEEE Infocom*, vol. 3, March 2005, pp. 1607–1617.

[7] D. S. Lun, M. Médard, and D. R. Karger, "On the dynamic multicast problem for coded networks," in *Proceedings of WINMEE, RAWNET and NETCOD 2005 workshops*, April 2005.

[8] M. Imase and B. M. Waxman, "Dynamic steiner tree problem," *SIAM Journal on Discrete Mathematics*, vol. 4, no. 3, pp. 369–384, August 1991.

[9] S. Raghavan, G. Manimaran, and C. S. R. Murthy, "A rearrangeable algorithm for the construction delay-constrained dynamic multicast trees," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 514–529, August 1999.

[10] [Online]. Available: www.cs.washington.edu/research/networking/rocketfuel