

MULTICAST TREE CONSTRUCTION IN DIRECTED NETWORKS

J. Eric Klinker

Center for High Assurance Computing Systems
Naval Research Laboratory
Washington, DC 20375
klinker@itd.nrl.navy.mil

Abstract

Significant interest exists within the military in moving towards an integrated services environment where traditional network services such as ftp, telnet, and e-mail can co-exist with real-time services such as voice, video, and satellite imagery. Multicast routing is an effective means of providing the efficient utilization of network resources required to realize such an environment.

Traditional multicast routing algorithms assume a symmetric network topology. Many military communication assets are either asymmetric in their load or asymmetric in capacity (a good example is Direct Broadcast Satellite). In addition, many military communication assets are bandwidth constrained, and routing symmetrically may further contribute to congestion. Therefore, multicast tree construction which tolerates network asymmetry is desirable for many military communication environments.

This paper proposes an algorithm for constructing shared multicast distribution trees in networks with asymmetric link capacities or loads. The algorithm tolerates asymmetry by building distinct, loop-free, sender and receiver paths onto a shared delivery tree. Additionally, the algorithm exhibits desirable security properties. Simulation results are presented that demonstrate the lower tree cost and better load balancing characteristics of the resultant trees over shortest path trees, with only a modest increase in path length.

Introduction

Current multicast routing protocols were developed under a paradigm that assumes network link symmetry. Many fixed networks and satellite or wireless networks exhibit asymmetry in capacity or load. In addition, traditional multicast routing algorithms utilize "reverse-path" routing which in asymmetric networks may lead to poor routes. Several studies [10, 3] have been made that confirm the existence of network asymmetry in the fixed networks that make up the internet

There are two basic approaches to multicast tree construction. The first is a shared multicast tree [1] and the other is a source rooted tree [13, 9]. The shared tree approach uses a single tree rooted at some center that is shared by all participants. In the source rooted approach, each sender builds a separate tree rooted at itself.

The algorithm presented in this paper builds a shared tree that can tolerate network asymmetry. To build this shared tree, the algorithm creates forwarding state in multicast capable routers. This state represents disjoint, loop-free, paths for senders and receivers.

The results of simulations conducted over varying topologies show that the resultant trees provide a lower tree cost than source rooted trees with only a modest increase in end-to-end delay.

Tree Construction

The join process is similar to the CBT approach [1] with some slight differences due to the asymmetry. A sender joins a tree by propagating a join-request message along the shortest path to the tree center. When the join-request reaches the center, a join-ACK message is sent back along the same path. A receiver joins in a similar manner by propagating a join-request to the center along the shortest path. The join-ACK is then sent back to the receiver along the shortest return path (which may be different than the path taken by the join-request). Since senders must explicitly join the tree (and thus the group) this violates the traditional multicast model of "senders just send"¹. However, this has some advantageous security properties in that sources can be authenticated before being admitted to the tree. The disjoint paths could result in routing loops if only group state is stored (as in CBT). Thus, to prevent routing loops the protocol must store source state as well, which has some scaling implications [7]. Thus, each router maintains a sender list (SL) per active group. The SL contains the following information [Sender ID, incoming interface, {set of outgoing interfaces (OGI)}].

The algorithm for adding a new sender is given below:

key

ξ_m	identifies the multicast group m
G	identifies the set of all multicast groups a router is aware of
$S_{m,i}$	identifies source i for multicast group m
$R_{m,i}$	identifies receiver i for multicast group m
N_i	identifies node i , a multicast capable router
N_{center}	identifies the node corresponding to the center of the multicast tree
SL_m	identifies a source list for multicast group m at a multicast capable router
$SL_{m[n]}$	identifies the n th element in the source list for multicast group m ,
MSL	represents the modified source list for a join-ACK packet
$I_{ogi,i}$	identifies the set of outgoing interfaces for source i in the SL
I_n	identifies the interface corresponding to the next hop towards a destination
I_p	identifies the interface corresponding to the previous hop of a packet
I_{prune}	identifies the interface a <i>prune</i> was received on
Rt.	an array that records the hop-by-hop route a packet has taken.

¹ A mechanism for non-member senders could be implemented by building wildcard sender (*,G) state. The set of outgoing interfaces for such state is simply the union of all other outgoing interface sets. Non-member senders, first send to the tree center. The data is then forwarded according to (*,G) state.

procedure Send_Join_Req($S_{m,i}, g_m$)

procedure Recv_Join_Req($S_{m,i}, I_p, g_m$)

```

begin
  if  $g_m \notin G$ 
     $SL_{m[0]} = [S_{m,i}, I_p, I_{center}]$ 
  else
     $SL_{m[n+1]} = [S_{m,i}, I_p, \{I_{ogi} \forall S_m \in SL_m\}]$ 
    for all  $I \in I_{ogi} \forall S_m \in SL_m$  do
      Send_Build_State( $S_{m,i}$ )
    end for
  end if
  if  $N_i \neq N_{center}$ 
    Send_Join_Req( $S_{m,i}, g_m$ )
  end if
end Recv_Join_Req

```

procedure Recv_Build_State($S_{m,i}, I_p$)

```

begin
  if  $S_{m,i} \notin SL_m$ 
     $SL_{m[n+1]} = [S_{m,i}, I_p, \{I_{ogi} \forall S_m \in SL_m\}]$ 
    for all  $I \in I_{ogi} \forall S_m \in SL_m$  do
      Send_Build_State( $S_{m,i}$ )
    end for
  else
    Send_S_Prune( $I_p, S_{m,i}$ )
  end if
end Recv_Build_State

```

procedure Recv_S_Prune($I_{prune}, S_{m,i}$)

```

begin
   $I_{ogi,i} = I_{ogi,i} \cap I'_{prune}$ 
  if  $I_{ogi,i} = \emptyset$ 
    Send_S_Prune( $I_{p,i}, S_{m,i}$ )
     $SL_m = SL_m - S_{m,i}$ 
  end if
end Recv_S_Prune

```

This algorithm is illustrated with the following example. Consider the topology in Figure 1. The multicast capable routers are lettered, and the links between them are numbered to identify specific interfaces. The solid arrows represent the current multicast delivery tree at the time the new sender wishes to join. The shaded arrows represent the path the join-request for the new sender (S3) takes to the tree center (router g). Build_State messages are generated at routers b and g and a prune must occur along the path [b, e, g]. Table 1. shows the state of the tree before the join-request, after all Build_State messages have reached their destinations, and after pruning is complete.

Figure 1. illustrates why the Build_State messages must be sent at each on-tree router. If the Build_State messages were sent only when the join-request reached the center, the Build-State message that reached router b would encounter the sender in the source list and the message would not get propagated out interfaces 3 or 4. This is eliminated if

Build_State messages are generated at router b as well as the center.

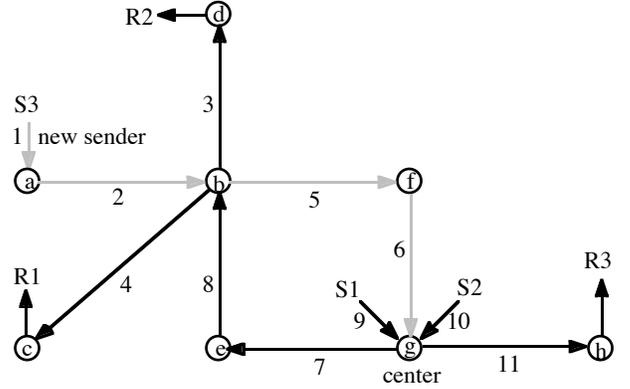


Figure 1. New Sender Example

Router	Current State	After Build-State Messages	Pruned State
a	No Senders	S3, 1, {2}	
b	S1, 8, {3, 4} S2, 8, {3, 4}	S1, 8, {3, 4} S2, 8, {3, 4} S3, 2, {3, 4, 5}	
c	S1, 4, {R1} S2, 4, {R1}	S1, 4, {R1} S2, 4, {R1} S3, 4, {R1}	
d	S1, 3, {R2} S2, 3, {R2}	S1, 3, {R2} S2, 3, {R2} S3, 3, {R2}	
e	S1, 7, {8} S2, 7, {8}	S1, 7, {8} S2, 7, {8} S3, 7, {8}	S1, 7, {8} S2, 7, {8} Remove S3
f	No Senders	S3, 5, {6}	
g	S1, 9, {7, 11} S2, 10, {7, 11}	S1, 9, {7, 11} S2, 10, {7, 11} S3, 6, {7, 11}	
h	S1, 11, {R3} S2, 11, {R3}	S1, 11, {R3} S2, 11, {R3} S3, 11, {R3}	S1, 9, {7, 11} S2, 10, {7, 11} S3, 6, {11}

Table 1. Adding a New Sender

The algorithm for adding new receivers is presented below:

procedure Center_Recv_Join_Req($R_{m,i}$)

```

begin
  for all  $\{S_{m,i} \in SL_m \mid I_n \notin I_{ogi,i}\}$  do
     $MSL = MSL + S_{m,i}$ 
  end for
   $Rt[0] = N_{center}$ 
  Send_Join_ACK( $MSL, R_{m,i}, Rt$ )
end Center_Recv_Join_Req

```

```

procedure Recv_Join_ACK(MSL,  $R_{m,i}$ , Rt)
begin
  if  $N_i \neq R_m$ 
    for all  $\{S_{m,i} \in MSL \mid S_{m,i} \notin SL_m\}$  do
       $SL_{m[n+1]} = [S_{m,i}, I_p, I_n]$ 
    end for
    for all  $\{S_{m,i} \in SL_m \mid S_{m,i} \notin MSL\}$  do
      if  $I_n \notin I_{ogi,i}$ 
         $I_{ogi,i} = I_{ogi,i} \cup I_n$ 
         $MSL = MSL + S_{m,i}$ 
      end if
    end for
    for all  $\{S_{m,i} \in MSL \mid S_{m,i} \in SL_m\}$  do
      Send_R_Prune( $S_{m,i}$ , Rt,  $R_{m,i}$ )
      if  $I_n \in I_{ogi,i}$ 
         $MSL = MSL - S_{m,i}$ 
      else
         $I_{ogi,i} = I_{ogi,i} \cup I_n$ 
      end if
    end for
     $Rt[n+1] = N_i$ 
    Send_Join_ACK(MSL,  $R_{m,i}$ , Rt)
  end if
end Recv_Join_ACK

```

```

procedure Recv_R_Prune( $S_{m,i}$ , Rt,  $R_{m,i}$ )
begin
  if  $|I_{ogi,i}| > 1$  or  $N_i = N_{center}$ 
     $I_{ogi,i} = I_{ogi,i} \cap I'_n$ 
  else
     $SL_m = SL_m - S_{m,i}$ 
     $Rt[n] = null$ 
    Send_R_Prune( $S_{m,i}$ , Rt,  $R_{m,i}$ )
  end if
end Recv_R_Prune

```

Consider the topology in Figure 2. Again the multicast capable routers are lettered, and the interconnecting links are numbered to identify interfaces. The solid lines represent the multicast delivery tree at the time the new receiver, (Ri) located at router f, is to be joined to the tree. The join-request propagates to the center and the join-ACK proceeds to the receiver along the path [a, b, c, d, e, f]. Shaded arrows represent links that require new state. The current state at each router is given in Table 2. As the join-ACK propagates towards the receiver, the state at each router is changed to the New State given in Table 2. The state of the Modified Sender List (MSL) is also shown in this column. When the join-ACK reaches router d it encounters S1 d's SL while S1 is also present in the MSL. Thus, for S1, a shorter path to router d exists and S1 must be pruned from the path traversed so far. The state of any router that is different as a result of the prune message is presented in the last column of Table 2. Some additional functions are required in the join-ACK to accommodate the pruning process. In order to determine the path along which to send the prune message, the join-ACK must record its route as it propagates to the receiver. This path

may also be discerned from the state information at each router (i.e. using the incoming and outgoing interfaces from

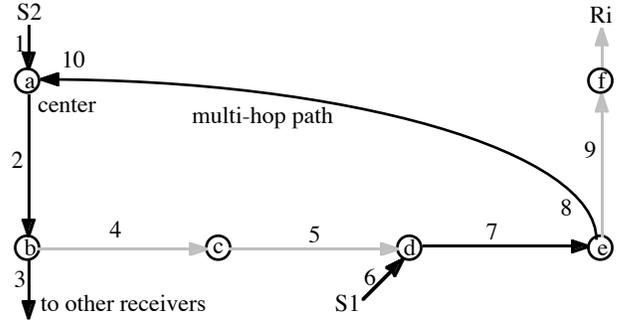


Figure 2. New Receiver Example

Router	Current State	New State	Pruned State
a	S1, 10, {2} S2, 1, {2}	No Change MSL = ϕ	
b	S1, 2, {3} S2, 2, {3}	S1, 2, {3, 4} S2, 2, {3, 4} MSL = {S1, S2}	S1, 2, {3} S2, 2, {3, 4}
c	No Senders	S1, 4, {5} S2, 4, {5} MSL = {S1, S2}	Remove S1 S2, 4, {5}
d	S1, 6, {7}	S1, 6, {7} S2, 5, {7} MSL = {S2}	Generate Prune
e	S1, 7, {8}	S1, 7, {8, 9} S2, 7, {9} MSL = {S1, S2}	
f	No Senders	S1, 9, {Ri} S2, 9, {Ri}	

Table 2. Adding a New Receiver

the sender list). When the prune message eventually encounters a set of OGIs that is larger than one it must determine which interface to prune from this set. Given the identity of the receiver, the interface corresponding to the next hop to the receiver should be pruned. Thus, the prune message must know the *sender id* that it is pruning and the *receiver id* it is pruning that sender for.

The tree construction protocol relies on several underlying mechanisms. The protocol uses the underlying unicast routing protocol to forward the control packets which construct the tree. It is assumed that Type of Service (ToS) routing can be performed to enforce policy during the tree construction. Tree construction also relies on protocols like IGMP [4] to reach a router that is group aware.

Since the protocol builds shared trees around a given center, some mechanism of informing participants of (group, center) mappings is required [2, 6]. An additional protocol which selects tree centers [12] is desirable as the quality of the shared tree will be highly dependent on the center selected.

Performance Evaluation.

Simulations were carried out to determine how the shared trees compared to the source based trees when reservation of resources was required. Resource reservation is required in any environment where bandwidth is constrained. Many military communication systems consist of low bandwidth links [8]. The algorithm was not compared to symmetric based algorithms since the difference in tree cost would merely be a reflection of the degree of asymmetry present in the network environment. The simulations were conducted to determine what penalties or benefits the shared trees incurred over their source based counterparts.

A random network was generated using Waxman's RG1 and RG2 algorithms [14]. Topologies consisted of several clusters, generated using RG2, that were meant to simulate separate routing domains. The clusters were interconnected with a network generated using RG1 and RG2. Topologies varied from 10-100 nodes distributed over 1-10 clusters. Link cost was defined as a uniform random variable between 1-100 for intra-cluster links and 1-1000 for inter-cluster links. Average node degree was kept in the interval [3,5] for each topology. The simulations also allowed several sessions to be built on top of each other so that load balancing over several sessions could be studied. The shared trees were constructed as per the protocols presented earlier. The shared trees were constructed around centers selected using the protocol given in [12]. The source based trees were constructed as the forward cost shortest path from the source to each receiver (much like MOSPF [9]).

Tree cost for each tree is calculated to represent the amount of resources that would have to be reserved over the entire tree for a specified number of concurrent senders. The reservation algorithm reserves the minimum amount of resources for any combination of concurrent senders. The tree cost is calculated by determining the number of senders that use each link on the tree. Let $S_{i,j}$ be the number of senders that use the link from node i to node j and $d_{i,j}$ be the cost to use that link. Let ss be the number of simultaneous senders. The cost for that link, $c_{i,j}$, is given by $c_{i,j} = \min(ss, S_{i,j})d_{i,j}$. The total tree cost, c_{tot} , is given by $c_{tot} = \sum_{i=1}^N \sum_{j=1}^N c_{i,j}$. The average path length is calculated as

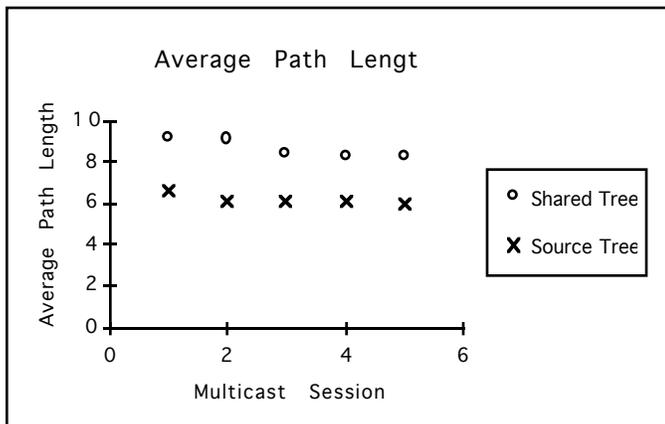


Figure 3. Average Path Length

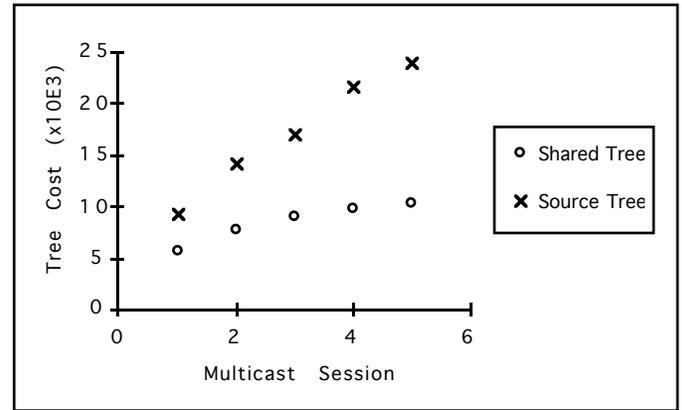


Figure 4. Tree Cost per Session

the average number of hops experienced by a sender to all receivers averaged over all senders. This is expected to be proportional to the delay experienced by a packet.

Figures 3, 4, and 5 show several characteristic results over a topology of 3 clusters each with 30 nodes. Five multicast sessions, each with 12 senders and 12 receivers evenly distributed over all domains, were constructed consecutively on top of each other. Figure 3 shows the average path length experienced by each sender per multicast session. As expected, the source based trees experience shorter paths. Figure 4 shows the tree cost per session, with 3 of the senders active. Up to 3 senders sharing a link from any previous session are considered when calculating tree cost. By observing the slope of the two plots it is evident that the shared tree distributes network load better over many sessions. Figure 5, the tree cost of each tree is examined as the number of concurrent senders is increased. The source rooted trees peak more rapidly then levels off, while the shared tree cost tends to increase as a constant rate. For larger topologies with larger numbers of concurrent senders it is possible for the cost of the shared tree to exceed the source tree. The source tree contains many links but these links are shared by only a few senders. The shared tree contains fewer links which are shared by most senders.

Figures 6 and 7 show representative results when the node degree of the topology is increased. As the node degree increases the cost of the source tree peaks more rapidly, and the crossover point for the shared tree is achieved sooner as

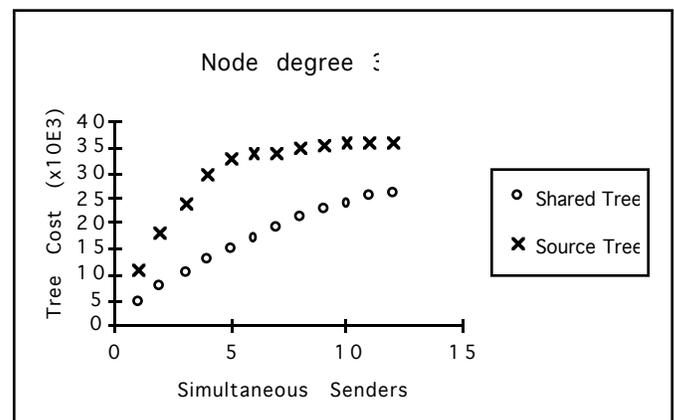


Figure 5. Tree Cost for Node Degree 3

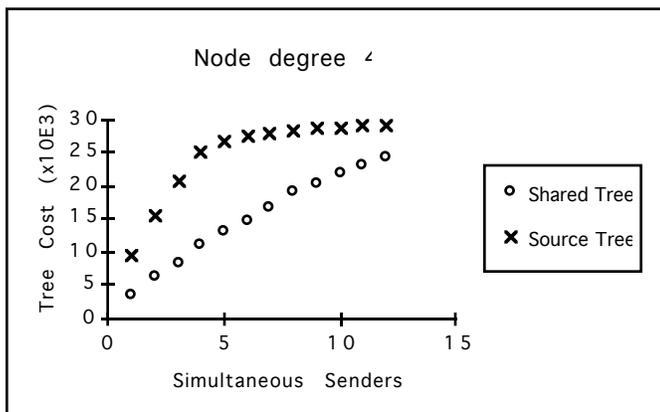


Figure 6. Tree Cost for Node Degree 4

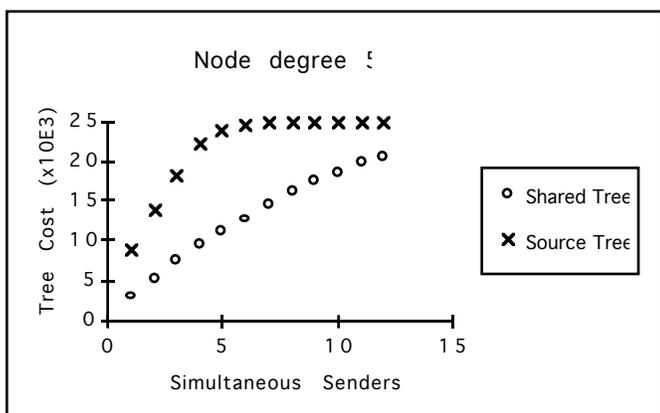


Figure 7. Tree Cost for Node Degree 5

the relative distance between the two curves is smaller.

Summary

For the number of nodes simulated, the results show that the shared trees do provide a lower tree cost than the source based trees with only a modest increase in delay.

However, the tree construction algorithm is not a complete multicast routing protocol. The protocol requires an additional mechanism to tear down state when participants leave or time out. Also several mechanisms should be borrowed from CBT [1] to maintain the tree in the event of failures and an additional ACK may be required to acknowledge the joining of a branch from center to receiver. The algorithm should also be optimized (e.g. consolidate some of the for loops).

The algorithm is expensive in state and thus, may not be appropriate for some military applications [7]. However, the size of the router state is comparable to PIM [5], the predominate commercial multicast routing protocol. Future work should involve extending the scope of the simulations. The simulator that was constructed has almost reached it's limit in terms of topology size and the size of the interactions it is capable of simulating. Many interesting military applications have characteristics outside the scope of the current simulation results. In addition, the results presented have interesting characteristics at the limits of the simulator.

References

- [1] A. Ballardie, P. Francis, and J. Crowcroft, "Core Based Trees (CBT) an architecture for scalable interdomain multicast routing", in *ACM SIGCOMM*, September 1993
- [2] B. Cain, S. Deering, and A. Thyagarajan, *Internet Group Management Protocol Version 3*, Work in progress.
- [3] K. Claffy, G. Polyzos, and H. W. Braun, "Traffic Characteristics of the T1 NSFNET Backbone," *Proc. of IEEE INFOCOM '93*, March 1993.
- [4] S. Deering, *Host Extensions for IP Multicasting*, RFC 1112, Network Working Group, August 1989.
- [5] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei, *Protocol Independent Multicasting (PIM)*, Work in progress.
- [6] M. Handley, J. Crowcroft, and I. Wakeman, *Hierarchical Protocol Independent Multicast (HPIM)*, Work in progress.
- [7] E. Klinker and S. Batsell, "The Implications of a Distributed Computing Paradigm on Multicast Routing", *Proc. of 1995 IEEE Military Communications Conf.*, November, 1995.
- [8] J. Macker, et. al., "QoS Requirements for Military Applications", Submitted to 1996 *IEEE Military Communications Conf.*
- [9] J. Moy, *Multicast Extensions to OSPF*, RFC 1584, Network Working Group, March 1994.
- [10] V. Paxson, "Growth Trends in Wide-Area TCP Connections," *IEEE Network*, August 1994.
- [11] S. Shukla, E. Klinker, and E. Boyer, "Multicast Tree Construction in Network Topologies with Asymmetric Link Loads," TR NPS-EC-94-012, Naval Postgraduate School, September, 1994.
- [12] S. Shukla, E. Klinker, and R. Voigt, "A Protocol for Locating Multicast Data Distribution Centers Using Participant Registration", TR NPS-EC-095-005, Naval Postgraduate School, February 1995.
- [13] D. Waitzman, C. Partridge, and S. Deering, *Distance Vector Multicast Routing Protocol*, RFC 1075, Internet Working Group, November 1988.
- [14] B. Waxman, "Routing of Multipoint Connections," *IEEE Selected Areas in Communications*, December 1988.