An On-Demand Shared Tree with Hybrid State for Multicast Routing in Ad Hoc Mobile Wireless Networks *

Elena Pagani, Gian Paolo Rossi
Computer Science Dept., Università di Milano, via Comelico, 39. I-20135 Milano, Italy
{pagae,rossi}@dsi.unimi.it

Abstract

This paper describes the Multicast Shared Tree Protocol *(*MSTP*) that has been designed to support the delivery of multicast packets in a mobile ad hoc network. MSTP uses a shared tree as its basic multicast infrastructure and, to maintain the tree against topology changes and network failures, it adopts the so called* hybrid-state *reliability mechanism, which combines the use of soft and hard state mechanisms applied to an on-demand scheme. It ensures fault tolerance and high group connectivity despite mobility, and provides a light-weight tree maintenance procedure.*

1. Introduction

This paper describes a routing architecture that efficiently supports the delivery of multicast packets in a mobile ad hoc network. The term *ad hoc* indicates a network in which the host mobility arbitrarily changes the network connectivity and where an infrastructure of base stations is not present [7]. In such a mobile environment, the multicast routing architectures that have been devised for wired networks cannot be adopted without modifications. Here, the challenge is to consider self adapting protocols which dynamically modify their behavior to fit the changing group requirements in terms of traffic patterns, membership and topological distribution.

We present the *Multicast Shared Tree Protocol* (MSTP) which has been specialized to address the case of, potentially many, small-medium sized groups of mobile hosts moving in a wireless space. MSTP uses a loop-free, shared tree to connect all the receiving hosts of a given multicast group. To ensure tree maintenance and reliability, MSTP adopts a *hybrid* mechanism which combines the use of soft-state and explicit reliability mechanisms and applies them to an on-demand scheme. This mechanism, which is called *hybrid-state*, ensures, at the cost of a negligible amount of control packets, flexibility and adaptability against topology changes, high multicast efficiency despite mobility, and a light-weight tree maintenance procedure which selectively intervenes on-demand only where damages occurred.

*This work was supported by the MURST under Contract no.9809321920, "Transport of multicast packets with QoS guarantees".

2. System and multicast model

The system we consider is composed of N mobile hosts, or hosts for short, with unique identifiers $p_1, \ldots, p_N$, that communicate via a packet radio network. We assume that all the mobile hosts have similar computing and storage capabilities.

MSTP does not depend on the network organization, that can be either flat or *hierarchical* (e.g., a clustered multi-hop network), and on the underlying wireless technology. For sake of concreteness, throughout the paper, we consider a hierarchical organization, obtained through a clustering protocol [6, 3]. In Figure 1, we show an example of such an architecture. Here, hosts are grouped into *clusters*; in a cluster, each host is one-hop neighbour of the clusterhead which coordinates the cluster communications among hosts. We indicate as one-hop *neighbors* a pair of hosts that can receive from one another. Clusters may be connected through gateways, such as the host p in Figure 1.

We consider a protocol stack in which the MAC protocol is layered on top of the clustering protocol; MSTP is constructed on top of the unicast routing protocol which, in turn, is constructed on top of the MAC protocol.

During the construction of the multicast tree, MSTP needs to know from the underlying unicast routing protocol "`which is my next-hop router to`

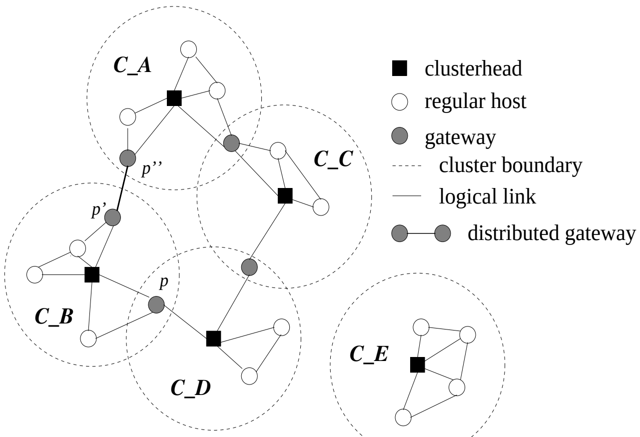

Figure 1. Example of a clustered multi-hop network.

`a given group destination`". It does not matter whether the answer comes from a classical distance vector table or from another routing policy which has been on purposely designed to address the mobility issues (e.g., [9, 2]). This makes MSTP independent on the unicast routing protocol, although, so far, simulations have been conducted by assuming an underlying distance vector routing protocol.

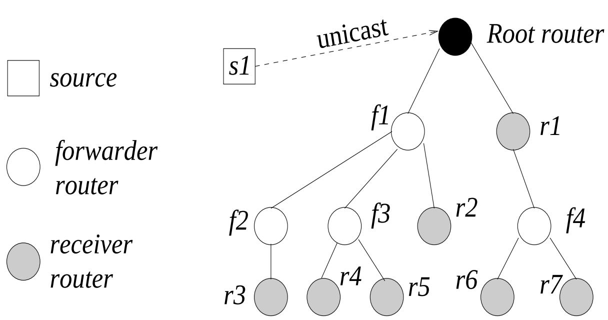

Figure 2. Example of a ST.

3. Background

MSTP uses the underlying service to construct a single shared tree ST (e.g., as the Core Based Tree, or CBT, protocol [1]) with center at the core host, henceforth called *Root*. The ST connects all the receiving hosts in a multicast group (Figure 2). Whatever way the *Root* is identified and placed (we do not face with this problem here [8]) its identity must be well known among the group members. This requirement can be easily achieved at configuration time or by contacting, at run time, a well known address where the *Root* name is published. As shown in Figure 2, all the network mobile hosts may eventually fall on the ST and perform routing functions. For this reason, we will often refer to them as *routers* to indicate that functionality of the hosts. The ST is composed of *receivers*, i.e. group members, and of *forwarders*, i.e. hosts that are not group members but are on the path from the *Root* to the members. The path from the *Root* to each group member is the *shortest* path provided by the underlying unicast routing, where 'shortest' indicates the less expensive path according to the cost metrics used by the unicast routing protocol. A multicast address is assigned only to the set of receivers in a multicast group [5]. That means that senders and receivers of a multicast group are separated and each host may become a sender for that group.

Whenever a source host has a packet to multicast, it uses the unicast addressing scheme to send the packet to the *Root* router. Once on the ST, the packet spans the tree on the basis of the group address. Receiver hosts may generate multicast messages as well. In this case, however, they will act as external source hosts by addressing unicast the *Root* which is the unique responsible of the multicast diffusion. This approach is motivated by the choice to deploy the message diffusion infrastructure that better fits with the unidirectional nature of most wireless links.

The choice of a shared tree for MSTP is motivated by the following arguments:

1. it offers good scalability and low cost operations when the network topology and the membership change;

2. the availability of a single *Root* node eases the interoperability with the fixed portions of an internetwork.

Nevertheless, the core node is prone to become a bottleneck when high data rate source applications are using the multicast tree. To address this point, MSTP can flexibly switch to a source-based structure under critical traffic conditions. The description of the switching mechanism is out of the scope of this paper [8].

MSTP adopts the so called *Hybrid-state* mechanism. The routers on the shared tree maintain a minimum set of information about the tree structure (as in the traditional *hard-state approach* [3]); namely, the identity of their up- and down-stream neighbours. The tree refresh mechanism can be selectively applied only to the damaged branches, while no overhead is generated to maintain the connected portions of the tree. The refresh is activated *on demand* whenever the *Root* has a packet to multicast. That means that the refresh period is dynamically regulated according to the data rate at the *Root*. Broken branches are removed by a *soft-state* mechanism [4], thus avoiding to generate control messages. MSTP tolerates link and host failures and recovers them by adopting a receiver-initiated recovery mechanism.

4. The protocol

In MSTP, the construction and the maintenance of the ST are initiated *on demand*, that is, when the *Root* has a packet to multicast, and are performed by the *Root*. Other shared tree implementations (e.g., [10]) provide distributed mechanisms to maintain damaged branches in which the recovery actions can be initiated by intermediate routers. In the current version of MSTP, we adopted a centralized control for two reasons. First, to simplify the mechanisms that ensure that trees are loop-free. Second, to avoid that routers on a damaged branch concurrently activate competing recovery procedures.

type `Build`	*source*	*G*	*TD_source.dwnstream*	*T_refresh*	*hop_count*

type `Lost`	*source*	*G*	*upstream*	*{set of lost destinations}*

Figure 3. Layout of the control packets.

As soon as the *Root* host has a message to multicast, it initiates the construction of the ST, that is the tree of the shortest paths to all destinations, on the basis of the underlying *unicast routing tables* (URTs for short). The ST is maintained according to the *hybrid-state* mechanism. If the host mobility does not affect the ST connectivity, ST can be further utilized for other messages; otherwise, the *Root* gets informed of the lost destinations and delays the maintenance of the tree until a new data packet arrives. Since we expect that the mobility rate is not uniformly spread over all the hosts, the tree construction applies only to the damaged branches, while the connected portions of the tree are reused. To this purpose, the protocol uses only two types of control packets: the `Build` packet, to build the tree or the tree branches, and the `Lost` packet, which is used to report upstream to the *Root* the unreachable destinations. The `Build` constructs or repairs the tree infrastructure before the diffusion of a multicast packet. The format of these control packets is given in Figure 3 for a group G. All the other actions undertaken to maintain the multicast tree are based on a soft-state strategy and do not require the use of extra control packets. In fact, the reception of a `Build` packet lights up the routing capabilities of a mobile host, which, from that time on, acts as an ST-forwarder for the time $T_{refresh}$. Should $T_{refresh}$ expire, a forwarder router will turn off its routing capabilities, while a receiver router enters a temporary state (COMPLAIN), in which it waits for a new `Build` packet to arrive (see, in Figure 4, the state transition scheme for both forwarder and receiver routers). In MSTP, the refresh mechanism, which prevents the ST-routers from being turned off, is provided by the multicast data flow. To this purpose, the *Root* router maintains an estimate of the data packets' arrival rate at the core. $T_{refresh}$ is computed accordingly and propagated to the ST's nodes through the `Build` and `multicast` packets. Time out can occur only when the *Root* estimate is wrong, or after link/host failures, i.e. some packet has been lost on the way to destination. In this event, a receiver-initiated recovery mechanism is activated towards the *Root* (COMPLAIN state).

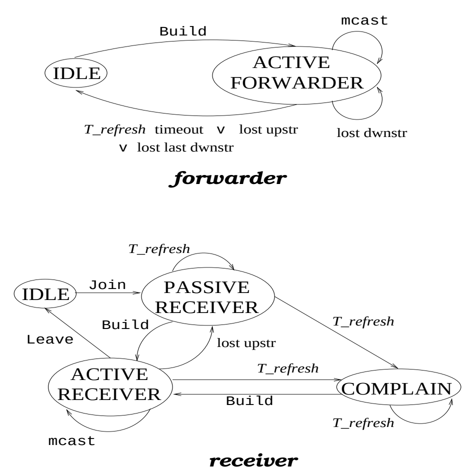

Figure 4. State transition schemes.

4.1. Tree construction without failures

In Figure 5, we report the data structures that are maintained by each router of the ST for a given group G and that are used to run the *Build-Tree* procedure. $IaminG$ allows a router to discriminate between forwarder and receiver functions. The use of the vector S is important at the *Root* router to dynamically list the set of receivers that need to be connected as soon as a `multicast` arrives. It initially contains the group membership, while at run time it contains the lost destinations. The *Tree Descriptor*, TD, describes, for a giver router r, the portion of the tree which is relevant for r, that is, the addresses of each one hop up and downstream routers and the list of receivers which can be reached through the tree with root in r. The function of the hop_count field in TD will be described in the next sections. The tree construction is initiated

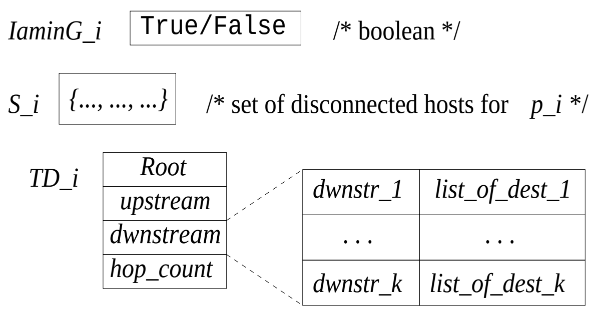

Figure 5. Data structures for host p_i.

by the $Root$ and continued by the hosts that are on the shortest path from the $Root$ to the destinations. Upon receiving a multicast packet from a source host, the $Root$ constructs the downstream entries in TD_{Root} by searching the local URTs for each destination in S. Then, it broadcasts $TD_{Root}.dwnstream$ within the `Build` packet. The hosts, belonging to the $Root$ cluster and indicated as next-hop router downstream to a set of G destinations, become active ST-routers for G. The other cluster's hosts discard the packet. The same procedure is recursively repeated by downstream routers until the tree spans to *leaf* routers. A leaf router is a router whose name is reported in the `Build` packet with an empty list of destinations.

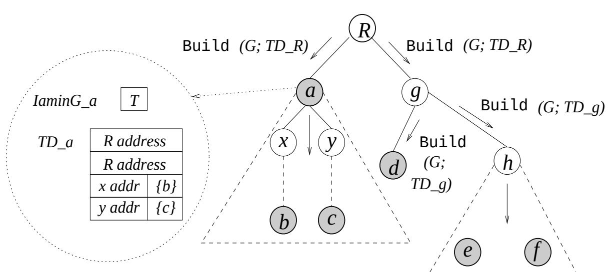

Figure 6. ST for hosts a, b, c, d, e and f, and GV_a.

In Figure 6, we show the ST that can be constructed by applying the above procedures to a group G in which hosts a, b, c, d, e, f are receivers.

4.2. Mobility management

When a host moves, the network topology changes and the ST should be modified accordingly. MSTP proceeds as follows. Each ST-router knows the name

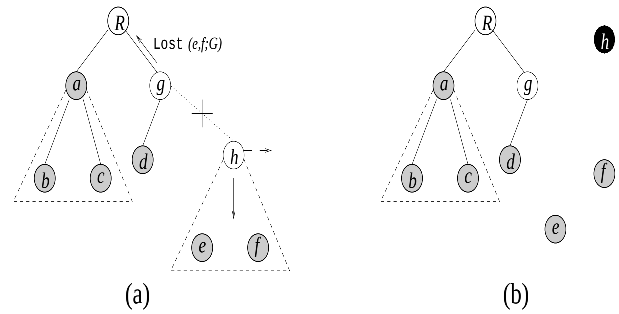

Figure 7. Host mobility.

of its one-hop neighbors by checking the local TD. As long as the up and the downstream routers are reported one-hop away, the ST connectivity is ensured. Let us consider the case of Figure 7, where host h is moving. The link from g to h of the ST is maintained while g sees h one-hop downstream. When this becomes false, g removes the entry for h in TD_g and sends the `Lost` packet upstream. The `Lost` packet flows upwards to report that receivers e and f have been lost and disconnected. Should the link between g and its upstream router (in this case the $Root$) be interrupted, g simply resets all local entries in the TD_g and goes back to the $idle$ state. In fact, when a forwarder router loses the upstream router it turns off the routing functions, while a receiver router enters the COMPLAIN state and maintains active the TD entries.

The mobility management procedure assumes that the upstream router will eventually observe the movement of its downstream router. This claim can be easily proved as follows [8]:

Claim 4.1 *In absence of failures, the Root knows the list of the disconnected receivers within at most the time t from their disconnection, where t is the time a* `Lost` *packet needs to traverse the height of the ST.*

If the 'loss' information about a disconnected receiver p_r is propagating upwards and, at the same time, the $Root$ is multicasting a packet downwards the ST, the host p_r will miss that packet. Only in this event MSTP cannot guarantee the full connectivity of the group membership. This represents a sort of shadow zone of the protocol which is in part emphasized by the fact that the tree maintenance is always initiated at the $Root$. This potential source of packet loss can be compensated with the introduction of a simple redundancy mechanism [8].

The use of $T_{refresh}$ eliminates the need of a pruning mechanism to prune downstream links of a broken branch. However, this way, *floating-subtrees* may be created. A floating subtree is a portion of the ST whose routers are unaware of having the branch interrupted somewhere upstream. Although the inconsistency will be promptly removed when $T_{refresh}$ expires, in the meanwhile floating routers may generate subtle loop conditions (see the next section).

4.3. Loop recovery

Loops may arise as the consequence of the mobility and of the slow convergency of URTs that are temporarily inconsistent. The ST construction presents three possible sources of loops that are described in Figure 8(a), (b) and (c). In the following, we only focus on the cases of Figure 8(a) and (b). A complete description of all loop recovery actions can be found in [8].

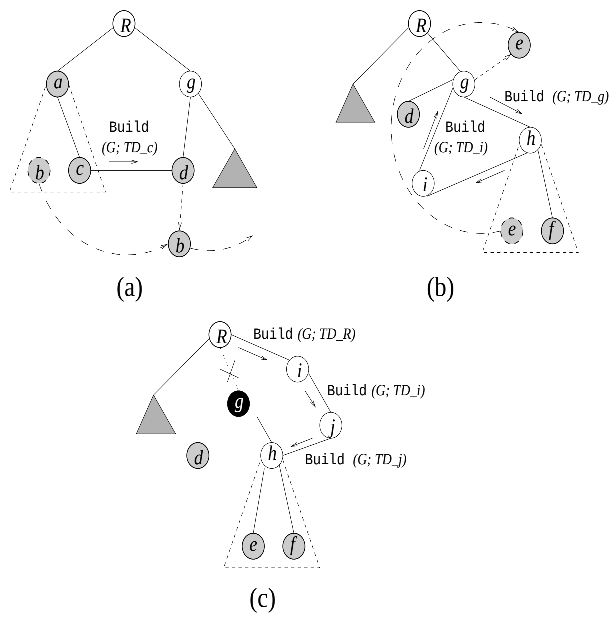

Figure 8. Loop conditions.

In Figure 8(a), host b moves from the subtree with root in a to the subtree with root in d. If the *Root* is sending the `Build`(b; G) packet while b is moving, it does so by routing it through node a according to its current URT. If the URT slowly reacts to b movements, a sub-optimal path can be created before b is caught up. The problem here is not the sub-optimality of the route, but the fact that, as shown in Figure 8(a), at a given point host c happens to report host d as its next downstream hop to b. d results to have two upstream parents and, in practice, it belongs to two separated branches.

In Figure 8(b) a similar event is shown in which a loop is generated by twisting the same branch: the *Root* sends a `Build` packet to connect host e. The `Build` packet is routed through the nodes g and h (the upstream of e). If host e is moving along the dashed trajectory of Figure 8(b), h tries to catch it up by forwarding the `Build` packet to i, which finally sends it to g, thus creating a new loop condition.

In both the above cases, a host that identifies a loop condition (d in Figure 8(a) and g in Figure 8(b)) must choose one of the competing branches and release the other. To this purpose, the `Build` packet transports the *hop_count* field, which is set to zero by the *Root*. When an ST-router receives the `Build` packet, it locally records the *hop_count* field and increments that value by one. When a `Build` packet is received from a different upstream router the branch reporting the lowest *hop_count* is selected and the other released (a `Lost` packet is sent upstream and the discussion of the previous section applies). This approach is motivated by the opportunity to privilege the shortest path between the *Root* and the considered ST router.

4.4. Link failures and packet recovery

So far, we assumed safe link transmission and infinite buffer space at the routers. Under these conditions, even if a router experiments a $T_{refresh}$ time-out, it knows that the *Root* estimate was wrong and a `Build` packet will eventually construct a new ST to connect it to the *Root*. When `Lost` and `Build` packets are lost or dropped while traversing the tree, a consistent view of the router state cannot be achieved and receiver routers can stay indefinitely waiting for a new ST branch to connect them. One possible solution to the problem is to adopt a soft state mechanism and periodically refresh the ST, thus allowing to clean up inconsistencies and to connect temporarily disconnected hosts. We have adopted a receiver-initiated approach which eliminates occasional flooding of membership packets and avoids the definition of a refresh rate. In the presence of failures, when $T_{refresh}$ expires at a host r, the best action r can undertake is to start complaining with the *Root* and stay waiting (a $T_{refresh}$ time) for some concrete event to happen, i.e. a `Build` packet that connects r to the ST. If nothing occurs, i.e. the communication with the *Root* is still faulty, r increases the level of complaining by shortening the retransmission time according to a decreasing function. Details about the recovery in a faulty environment can be found in [8]. The complain has the form of a *bubble* packet, i.e. an empty data packet sent unicast to the *Root* and carrying the $T_{refresh}$ value for r. Upon receiving the *bubble*, the *Root* in autonomy decides whether to reply with a '`relax`' message, i.e. it acknowledges the *bubble* by updating the current *Root*'s $T_{refresh}$ estimate, or to send a `Build` packet. Even in a faulty environment, it is still easy to prove the claim 4.1. In this case, the time t spans over a larger range whose estimate is informally given in [8].

In [8] we also prove the following claim:

Claim 4.2 *If eventually the network topology stabilizes, and transient link failures are recovered within a finite number of retransmissions, then: (i) within a finite time each destination in the* Root *partition is connected to the* Root*; and (ii) the resulting structure is acyclic.*

5. Concluding remarks

This paper describes the MSTP Protocol that has been designed to provide an efficient multicast delivery in mobile wireless networks. There are three main design choices being adopted in MSTP to address the considered mobility requirements: 1) it utilizes a shared tree; 2) it utilizes the hybrid-state approach to maintain the tree infrastructure; 3) it applies them to an on-demand scheme. As the consequence, MSTP provides its service at the cost of a negligible control packet traffic, while a certain, although limited, amount of routers memory overhead has to be payed.

We measured the performance of MSTP by means of simulations, and we compared it with a soft-state algorithm under different traffic and mobility conditions by measuring the *multicast efficiency* [4], i.e. the ratio between the total number of hops of multicast packets and the total number of multicast transmissions, and the *throughput*. Simulation results are highly encouraging [8] and show that the hybrid-state approach guarantees a good bandwidth utilization and a high multicast efficiency with respect to the soft-state approach. Both the throughput and the efficiency are negatively affected by the speed of mobile hosts, while the performance loss is independent of the group cardinality. In fact, the structure built with the soft-state approach is characterized by a high amount of redundant paths. This negatively affects the efficiency and the throughput, because of a higher collision probability. By contrast, it ensures a high degreee of reachability of the group destinations and a high stability of the multicast structure despite mobility. According to the measured results, the hybrid-state approach performs better than soft-state when the movement speed is less than 70-80 Km/hr and when the group cardinality does not exceed 50/60 members. As the consequence, the hybrid-state does not scale well to large groups where performances are negatively affected by the tree depth. This can be easily controlled, but at the cost of a further memory overhead. We are currently working to a new version of MSTP which adopts a meshed multicast infrastructure to solve the scalability issues and allows intermediate routers to initiate the tree repair procedures.

References

[1] Ballardie T., Francis P., Crowcroft J. "Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing". *Proc. SIGCOMM'93*, pages 85–94, Sep. 1993.

[2] Broch J., Maltz D. A., Johnson D. B., Hu Y.-C., Jetcheva J. "A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols". *Proc. ACM/IEEE International Conference Mobicom'98*, Oct. 1998.

[3] Chiang C.-C., Gerla M. "Routing and Multicast in Multihop, Mobile Wireless Networks". *Proc. IEEE ICUPC'97*, Oct. 1997.

[4] Chiang C.-C., Gerla M., Zhang L. "Forwarding Group Multicast Protocol (FGMP) for Multihop, Mobile Wireless Networks". *ACM/Baltzer Journal of Cluster Computing: Special Issue on Mobile Computing, Vol. 1, No. 2*, 1998.

[5] Deering S. E. "Multicast Routing in a Datagram Internetwork". *Ph.D. dissertation, Stanford University*, Dec.1991.

[6] Gerla M., Tzu-Chieh Tsai J. "Multicluster, Mobile, Multimedia Radio Network". *ACM/Baltzer Journal of Wireless Networks, Vol. 1, No. 3*, pages 255–265, 1995.

[7] Macker J. P., Corson M. S. "Mobile Ad Hoc Networking and the IETF". *ACM Mobile Computing and Communications Review, Vol. 2, No. 4*, pages 9–13, Oct. 1998.

[8] Pagani E. "Primitives for Reliable Group Communication in Distributed Systems with Mobile Hosts". *Ph.D. dissertation, Dip. di Scienze dell'Informazione, Università degli Studi di Milano*, 1998.

[9] Perkins C., Royer E. M. "Ad Hoc On Demand Distance Vector (AODV) Routing". *Internet Draft, draft-ietf-manet-aodv-01.txt*, Aug. 1998. Work in progress.

[10] Ramanathan R., Steenstrup M. "Hierarchically-Organized, Multi-hop Mobile Wireless Networks for Quality-of-Service Support". *Journal of Mobile Networks and Applications, Special Issue 'Mobile Multimedia Communications', Vol. 3, No. 1*, pages 101–119, Jan. 1998.